BASIC MANUAL
FOR
HITACHI PERSONAL COMPUTER
MB-S1 10/20

This manual is a translation of the manual for Personal computer MB-S1 10/20 for use in Japan.
Some of the specifications may be changed in export models.

PREFACE

This manual describes the syntax of the BASIC language used by the MB-S1 Series Hitachi Personal Computer. In addition to the Basic Master Level 3 BASIC, the MB-S1 Series is provided with S1 BASIC, which is a higher version of Level 3 BASIC, and this manual explains both BASICs.
For details of the hardware, please refer to the operation and maintenance manual of the related MB-S1 series computer.

CONTENTS

CHAPTER 3. FUNCTIONS

APPENDIX

CHAPTER 1. OUTLINE OF BASIC

1.1 OPERATION MODES

There are two types of BASIC operation mode; the direct mode and the program mode. When BASIC is started (refer to the operation manual), the title of the BASIC is displayed on the screen and after that "READY" is displayed. In this state, BASIC instructions (commands and statements) can be input.

Direct mode

When a BASIC instruction is input without a Line No. and the return key is pressed, the instruction is immediately executed. This is called "execution of an instruction in the direct mode".
When MON ↵ (return key) is pressed while the computer is in waiting for instruction input, the monitor mode is set. When TERM ↵ is pressed, the terminal mode (communications mode) is set.
In the monitor mode, the state of memory can be referred to or altered by using an instruction called a "subcommand", and this function is very effective in debugging machine language programs.
In the terminal mode, the MB-S1 can be used as a terminal of a large computer or it can be connected to another personal computer via a communications line.

Program mode

When an instruction is input with a Line No. (0 ~ 63999) while the computer is waiting for instruction input, the instruction is not executed but is stored in the memory as a program line. After storing a number of instructions with a Line No. for each, if a RUN command, GOTO statement, or GOSUB statement having no Line No. is executed, the stored program is executed. This is called "execution in the program mode".
When the program ends, the computer returns to the "waiting for instruction input" state.

1.2 LINE AND LINE NO.

The area starting from the input of a character and ending at pressing of the $\boxed{\lrcorner}$ key is called a line of BASIC. The length of a BASIC line is up to 255 characters. No character exceeding 255 can be input. The line is structured as shown below:

```
nnnnn␣ BASIC instruction [:BASIC instruction ...] ['comment]
```

←——— within 255 characters ———→
(255 bytes)

- ┌nnnn┘ shows the Line No. Specify an integer value in the range 0 to 63999. Line numbers are used to register the sequence of program execution and the execution starts from the smallest Line No. and sequentially moves to larger Line Nos. When a Line No. is omitted, the corresponding line is executed in the direct mode and is not registered as a program.
- ┌␣┘ shows a blank area of one or more characters. ┌[ ]┘ shows that the description inside the brackets can be omitted.
- Two or more instructions of BASIC can be input in one line, and ┌:┘ (colon) is used to delimit instructions. (This is called a multi-statement.)
- A comment may be input with ┌'┘ (single quotation mark) written after an instruction and before the comment. The description after ┌'┘ is ignored at the time of program execution.

Sometimes, ┌.┘ (period) can be used instead of a Line No. A period represents a line stored by BASIC at a time of error or editing, and it can be used in such instructions as LIST, DELETE or EDIT.

(Example)	LIST.	The line stored by BASIC is displayed. This is especially helpful to display the line where an error occurred at the time program execution stopped because of the error.
	DELETE. –500	Deletes all lines between the line in the BASIC and Line No. 500.

1.3 USABLE CHARACTERS

Alphabetic characters, numerics, KANA characters and special characters can be used in BASIC.

- Alphabetic characters from A through Z, both capital and small characters, can be used.
- Numerics are from 0 through 9.
- There are two types of special characters; special symbols and graphic characters. The meaning of each special symbol is shown below.

Symbol	Name	Meaning
$\boxed{\ }$	Space or Blank	As a rule, BASIC ignores a space. If, however, it is defined as a character constant (refer to 1.5 Constants), it has meaning as a character.
$\boxed{=}$	Equal	Used in assignment statements and relational operators.
$\boxed{+}$	Plus	Used as a positive code or for addition.
$\boxed{-}$	Minus or Hyphen	Used as a negative code or for subtraction.
$\boxed{*}$	Asterisk	Used as a multiplication sign.
$\boxed{/}$	Slash	Used as a division sign.
$\boxed{\text{¥}}$	Yen	Used as an integer division sign.
$\boxed{\wedge}$	Up arrow	Used as an exponential sign.
$\boxed{(}$	Left parenthesis	Used to specify the operation priority order.
$\boxed{)}$	Right parenthesis	Used to specify the operation priority order.
$\boxed{\%}$	Percent	Used in variable statements of variables.
$\boxed{\#}$	Sharp or Number	Used in double precision statements of variables.
$\boxed{\$}$	Dollar	Used in character statements of variables.
$\boxed{!}$	Exclamation mark	Used in single precision type statements of variables.
$\boxed{\&}$	Ampersand	Used in octal or hexadecimal constant statement. (Example) Octal number . . . . &01234 Hexadecimal number . . . . . &H5FA3
$\boxed{@}$	At mark	No special meaning
$\boxed{,}$	Comma	Used as a delimiter of parameters used in an instruction.
$\boxed{.}$	Period	Used as a decimal point or as a pointer of the line in BASIC memory. It is also used as abbreviations of instructions.
$\boxed{'}$	Single quotation mark	Used as a substitute of a REM statement.

Symbol	Name	Meaning
$\boxed{;}$	Semicolon	Used as a delimiter of PRINT or other statements.
$\boxed{:}$	Colon	Used as a delimiter of a multi-statement.
$\boxed{?}$	Question mark	Used as a delimiter of a PRINT statement.
$\boxed{<}$	Less-than sign	Used as a relational operator.
$\boxed{>}$	Greater-than sign	Used as a relational operator.
$\boxed{\text{"}}$	Double quotation mark	Used to define a Hollerith constant.

1.4 SCREEN EDITOR

The screen editor is the function that prepares, adds or deletes programs on the screen. Here, the use of screen editor is explained by actually inputting a short program.

Program preparation

First, execute the NEW command and erase the program in the memory. Then, input the next program (the color of the whole screen will changed while music is being output). Either upper or lower case alphabetic characters can be input. Press the [⏎] key each time a line is input. These lines are stored in memory as a program.

```
NEW[⏎]                                    ............ Erases the program on the screen.
10␣BEEP[⏎]                                ............ Emits by beep.
20␣CLS[⏎]                                 ............ Clears the screen.
30␣PAINT␣(320, 100), 1[⏎]                 ............ Displays a blue color over the en-
40␣PLAY␣"T80O4CDEFGABO5CMB"[⏎]............             tire screen.
                                                       Plays music 「do, re, mi ...」.
50␣FOR␣I=1␣TO␣8[⏎]     )
60␣PALETTE␣1, I[⏎]     }                  ............ Changes the color.
70␣NEXT␣I[⏎]           )
80␣END[⏎]
```

(Note) 「␣」 is a blank. It does not cause any display on the screen, which is the same result as pressing [⏎].

Correction of a program

After inputting a program, check the contents using the LIST command. If there is an error in the program, correct it in the line units. Make sure to press the ⏎ key after the correction.

```
LIST ⏎
10 BEEP
20 CLS
30 PAINT (320,100),1
40 PLAY "T80O4CDEFGABO5CMB"
50 FOR I=1 TO 8
60 PALETTE 1, I
70 NEXT I
80 END
Ready
```

(1) **Character correction**

If an incorrect character has been input, move the cursor to the position of character to be corrected by pressing the cursor movement keys (↑ ↓ ← →), and input the correct character.

(2) **Character adding**

There are two ways to add a character as explained below.

- Using the INS key

  Move the cursor to the character next to the character position in which a character is to be added and press the INS key. The character at the position of the cursor is reversed and a character can be inserted. When the character to be added is input, it is inserted before the reversed character. Press the ⏎ key or press the INS key once more. The character insertion condition is released and the cursor returns to the normal state.

- Using the IS/BS key

  Move the cursor to the character next to the character position in which a character is to be added and key in SHIFT + IS/BS . Then, a blank area for one character is generated before the character in the position of the cursor. Key in SHIFT + IS/BS for the number of times equivalent to the number of characters to be added, and then insert the new characters.

(3) **Character deletion**

There are two ways to delete a character.

- Using the DEL key

  Move the cursor to the position of character to be deleted and press the DEL key. The character is deleted and the characters after that are shifted forward.

- Using the IS/BS key
  Move the cursor to the character next to the character to be deleted and press the IS/BS key. Then, the character before the cursor is deleted and all characters after that are shifted forward.

Adding lines

When the program has been corrected, execute it. Input the RUN command. 「do, re, mi...」 is played but the screen color cannot be identified. This is because the processing of lines 50 to 70 is too fast for the human eyes to recognize. In order for the color change to be displayed slowly, insert the next program lines between Line No. 50 and Line No. 60.

```
55␣FOR␣J=1␣TO␣500 ⏎
56␣NEXT␣J ⏎
```

When adding lines, use numbers other than Line No. already used for the lines to be added. If there are many lines to be added or if new line numbers are not available, rearrange all the line numbers before adding the new lines.
Execute the LIST command and make sure that the program has been corrected.

```
55 FOR J=1 TO 500⏎
56 NEXT J⏎
LIST⏎

10 BEEP
20 CLS
30 PAINT (320, 100), 1
40 PLAY  "T80O4CDEFGABO5CMB"
50 FOR I=1 TO 8
55 FOR J=1 TO 500       } ..... Lines added
56 NEXTJ
60 PALETTE 1, I
70 NEXT I
80 END

Ready
```

After checking the program correction, execute the program with a RUN command. This time, the music of 「do, re, mi ...」 is played while the color of the screen changes.

Rearrangement of line numbers

Some of the line numbers are now close together as a result of adding new lines. Execute the RENUM command and then check the line numbers by executing the LIST command.

```
RENUM 100↵
Ready
LIST↵

100 BEEP
110 CLS
120 PAINT (320, 100), 1
130 PLAY  "T80O4CDEFGABO5CMB"
140 FOR I=1 TO 8
150 FOR J=1 TO 500
160 NEXT J
170 PALETTE 1, I
180 NEXT I
190 END
```

The lines numbers are now in steps of 10 starting at 100. For details, refer to the RENUM command.

Ready

Deletion of lines

(1) **Deletion of a line**

Input the number of the line to be deleted and press the ↵ key. Try deleting the END statement on the last line as an example.

```
190↵
LIST↵

100 BEEP
110 CLS
120 PAINT (320, 100), 1
130 PLAY  "T80O4CDEFGABO5CMB"
140 FOR I=1 TO 8
150 FOR J=1 TO 500
160 NEXT J
170 PALETTE 1, I
180 NEXT I
                ←————————————— Line No. 190 is deleted.
Ready
```

(2) **Deletion of two or more lines**

Execute the DELETE command to delete two or more consecutive lines. In the following example, the lines changing the color are deleted.

```
DELETE 140–180[↵]

Ready
LIST[↵]

100 BEEP
110 CLS
120 PAINT (320, 100), 1
130 PLAY "T80O4CDEFGABO5CMB"

Ready
```

Copying a line

When the lines with the same contents must be used in a program, the second line can be added by copying the first line and assigning a new Line No. to the copied line. In the following example, the first BEEP statement is repeated at the end.
Move the cursor to the Line No. of BEEP statement, change the Line No. to 500 and press the [↵] key.

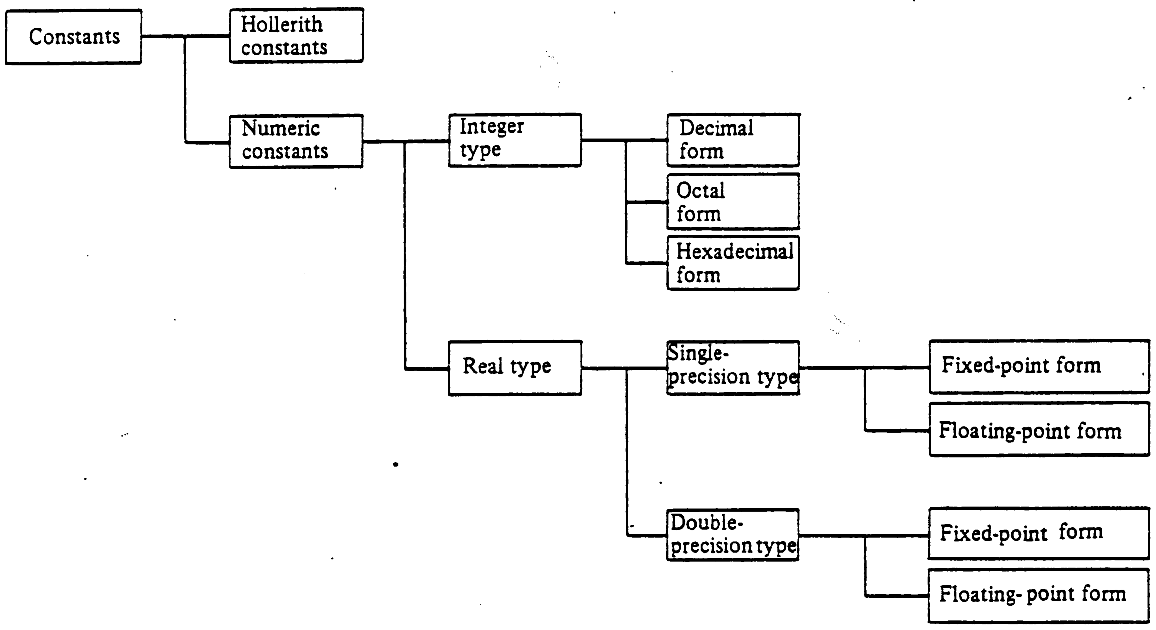
```
[500][↵]BEEP              100 is changed to 500.
110 CLS
120 PAINT (320, 100), 1
130 PLAY "T80O4CDEFGABO5CMB"
```

Although the display on the screen does not change, the program in the memory is changed. Check this with a LIST command. (The BEEP statement on Line No. 100 should remain as it is and the same contents should be on Line No. 500.)

```
LIST[↵]

100 BEEP
110 CLS
120 PAINT (320, 100), 1
130 PLAY "T80O4CDEFGABO5CMB"
500 BEEP

Ready
```

1.5 CONSTANTS

The following constants are used by BASIC.



Hollerith constant

When using a specific character string in a program, enclose the character string between two 「"」 characters. This is called a Hollerith constant. The length of such a character string can be up to 255 characters. A character string having a length of "0" is called a null string. If 「"」 itself is to be used as a part of character string, the string must be enclosed between two CHR$ (34).

(Example) " HELLO "
" HIRAGANA KATAKANA123 "
CHR$ (34) + " HITACHI " + CHR$ (34)
" " .... This is a null string.

Numeric constant

There are two types of numeric constant; integer type and real type, and a constant of either type can have a positive, negative or "0" value. If it has a negative value, a minus sign (−) must be put in front of it. It is has a positive value, the plus sign (+) may be omitted. Use of a comma (,) within a numeric constant is prohibited.

Integer type

(1) **Decimal form**

This is the same as an integer used in arithmetic, but usable numbers are limited to the range from –32767 to 32767. Any number outside of this range, even if it is an arithmetic integer, is handled as a constant of the single-precision or double-precision type. The fact that it is an integer may be shown by writing 「%」 (attribute character) after the number.

Using a decimal point is not permitted.

(Example) 32767
–6324
123% ..... Shows that this is an integer.

(2) **Octal form**

An octal number is expressed by writing 「&」 or 「&O」 at the start of a numerical value consisting of numerics "0" to "7". A sign may be written before 「&」 or 「&O」. The usable numbers are in the range from –&O177777 (–65535) to &O177777 (65535).

(Example) &123
–&O1777

(3) **Hexadecimal form**

A hexadecimal number is specified by writing 「&H」 at the start and it is expressed using "0" through "9" and "A" through "F". A sign may be written before 「&H」. The correspondence between decimal numbers and hexadecimal numbers is as shown below.

Hexadecimal No.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Decimal No.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Usable numbers are in the range from –&HFFFF (–65535) to &HFFFF (65535).

(Example) &H2AF0
–&H1A

Real type

Real numbers can be classified into single-precision type and double-precision type number from the viewpoint of precision, fixed-point form and floating-point form from the viewpoint of the way they are expressed.

(1) **Single-precision type**

A constant having a value in outside the integer type range or having a fractional part is treated as a number of single-precision or double-precision type. In such a case, if the number of significant digits is up to seven and the absolute value is in the range from about 6.0×10^{-39} to about 8.5×10^{37} or if the value is "0", the constant is handled as a single-precision type constant. All single-precision type constants are handled with the precision of seven digits and up to six digits are displayed. Attribute character 「!」 may be written after the number to distinguish it. For explanation of a case in which the calculation result becomes different by having or not having 「!」 written after a number, refer to (4) Floating-point form.

The following summarizes the conditions that a numeric value is regarded as a number of the single-precision type.

■ A real number consisting of up to seven digits	(Example) 54.3
■ When accompanied by an attribute character 「!」	(Example) 243 !
■ Exponential form using 「E」	(Example) −9.8E−4

(2) **Double precision type**

A constant having eight or more up to 16 significant digits and an absolute value in the range from about 6.0×10^{-39} to about 8.5×10^{37} or "0" is a number of the double-precision type. If such a number is used as a constant, it is automatically handled as a number of the double-precision type as long as it consists of eight or more significant digits. When eight or more digits are wanted in the result of an operation with a number having less than eight digits, the distinction can be made by writing attribute character 「#」 after the number. For example, when the following is input:

PRINT 10/7 ..... 「/」 is a division operator 「÷」.

Both "10" and "7" are calculated in single precision since they have less than eight digits, and the result is displayed as "1.42857". On the other hand, when the input is done as follows:

PRINT 10#/7#, or PRINT 10.000000/7#,

"10#" and "10.000000" are regarded as numbers of the double-precision type and the calculation is done in double precision, displaying the result as "1.428571428571429". To use double precision calculations, specify all constants as double-precision.

(Example)

```
PRINT 0.1#/0.3#
  .3333333333333333
```

Both the calculation and display are conducted in double precision

```
PRINT 0.1#/0.3
  .3333333200878573
PRINT 0.1/0.3#
  .3333333383003871
```

Although the display is of the double-precision type, the calculation result is in single precision. (Either the denominator or numerator is single precision.)

The following summarizes the conditions for a numeric constant to be of the double-precision type.

1	Eight or more significant digits	(Example) 123456789
2	A number accompanied by attribute character 「#」	(Example) 4.56#
3	Exponential form using "D"	(Example) 1.23D 5

(3) **Fixed-point form**

The expression form consisting of numerals "0" through "9" with "+" or "−" (the "+" sign may be omitted) and a decimal point is called the fixed-point form.

(Example) −1.25
.56789046

(4) **Floating-point form**

The expression of a real number containing a fraction in the form "A x 10^n" and showing the "A" value (called the mantissa) and "n" value (called the exponent) is called "floating point". The mantissa must be expressed in the fixed-point form and the exponent must be an integer of no greater than two digits.

The floating-point form in the single-precision type is expressed using character "E" after the mantissa which it is followed by the exponent. The floating-point form of a double-precision type number is expressed using character "D" instead of "E".

(Example)

```
PRINT 1.2345E−2
 .012345
PRINT −2.3476E10
−2.3476E+10                   ...... Single-precision type

PRINT 1289E4
 1.289E+07

PRINT 1.2345D−2
 .012345

PRINT −2.3476D10              ...... Double-precision type
−23476000000

PRINT 1289D4
 12890000
```

Note that the form in which the results of the fifth and sixth examples is expressed is different from that of the single-precision type. On a computer display, if the number of digits to be displayed is not greater than the maximum number of effective digits of the form, the display is made in the fixed-point form, and if it is displayed in the floating-point form, the exponent part is always displayed as a sign and an integer of two digits. Also, the integer part of the mantissa part is always one digits.

In the case of the floating-point form, since the single and double precision types can be clearly distinguished, there is no need to use an attribute character like 「!」 or 「#」.

(Example) 5.63E 4

−2.11D−3

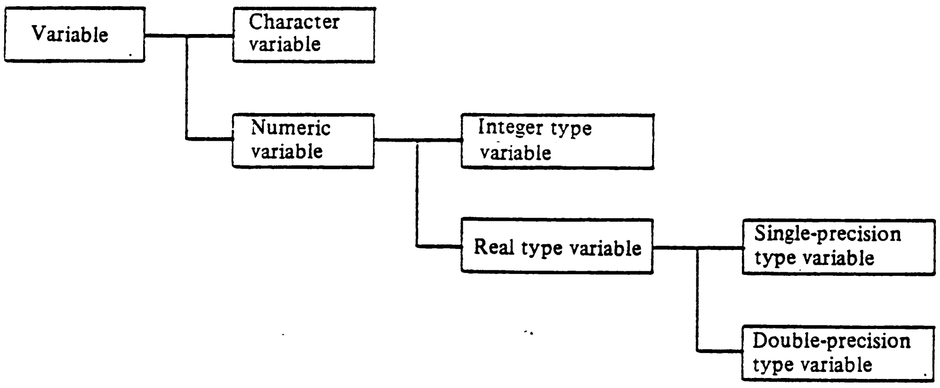
The following shows examples in which the calculation result and display are different depending on the use or non-use of 「!」 or 「#」. 「;」 is a sign used in a PRINT statement in which numerals or character strings are displayed in succession. For further details, refer to the explanation of PRINT.

(Example)

```
■ PRINT 12345678/3; 12345678!/3
   4115226  4.11523E+06
■ PRINT 987654 * 71; 987654 #* 71 #
   7.01234E+07  70123434
■ PRINT 1/3; 1 #/3 #
   .333333  .3333333333333333
■ PRINT 2.000004; 2.000004 #; 2.0000004
   2  2.000004  2.0000004
```

1.6 VARIABLES

The variable is a place in which character strings or numeric values used in a BASIC program are stored and each of them is given a name for to distinguish between them. A name given to a variable is called a variable name.
BASIC uses the following variables.



In addition to the above classification, variables can be divided into two types, simple variables and array variables.
In the case of a simple variable, one memory location corresponds with one variable name and the contents of the memory location can be referred to by specifying the variable name.
An array variable is a group of two or more data items having the same nature and two or more elements can be referred to by specifying a variable name. Each of these elements can be specified by assigning a subscript.

Variable names and type statements

Variable names are subjected to the following restrictions:

1) A variable name must be expressed with an alphabetic character at the start followed by alphabetic character(s), numeric character(s) and attribute character (sometimes this is not needed). Either capital or small alphabetic characters can be used. (If a small character is input, it is converted to a capital character internally.)
2) The length of variable name is up to 255 characters, but distinction is made by the first 16 characters and attribute character only. In other words, when a variable name that is longer than 16 characters is used, if there is another variable name having the same first 16 characters and attribute, the two are handled as the same variable.

3) The first word of a variable must not be a reserved word of BASIC (keyword used in commands, statements and delimiters, refer to the list of reserved words).

Example of correct variable name:	ABC	
	IDATA1	
Examples of incorrect variable name:	1DATA	Starts with a numeric.
	DATA1	Starts with a reserved word (DATA)
	TOWARD	Starts with a reserved word (TO)

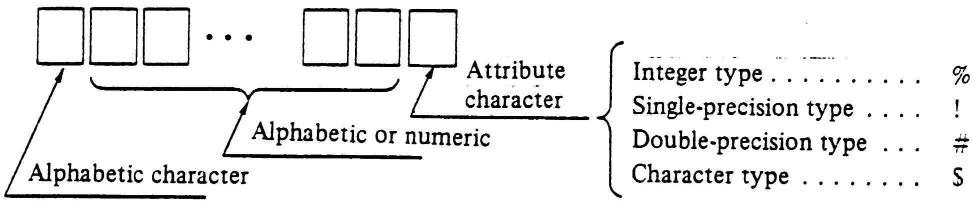
An attribute character used in a variable name shows the type of value which can be stored in the variable, and it is written at the end of the name. Variables have the same two types, numeric and character, as constants, and numeric variables can be classified into integer type variables, single-precision type variables and double-precision type variables. The table belows shows attribute characters used in variables names and the corresponding types.

Attribute character	Type	Size of storage of variable value (Note 1)	Example
%	Integer type	2 bytes	A% APL%
!	Single-precision type	4 bytes	B! SV!
#	Double-precision type	8 bytes	C# LEVEL #
$	Character type	0 ~ 255 bytes	D$ ALPHA $
None	Single-precision type (Note 2)	4 bytes	E XLE CCK

(Note 1) The storage size does not include the space occupied by the variable name.

(Note 2) The declared type is taken when a variable names starts with an alphabetic character specified by a DEFINT/DBL/STR statement.

To summarize the above explanation, the format of variable names is as shown in below:


Attribute character
Alphabetic or numeric
Alphabetic character
Integer type .... %
Single-precision type .... !
Double-precision type ... #
Character type .... $

In BASIC, a variable name is registered by storing it in the internal memory.
The space occupied in memory by a variable is automatically secured in the user memory area by BASIC from the first use of the variable. The size of the memory space depends on the variable type as shown in the table above.
In the following cases, the memory area secured for variables is released and the contents of all variables are erased.

a) When a CLEAR instruction is executed.
b) Immediately before executing a program using a RUN command.
c) At the time of executing a NEW command. (The program is also erased.)
d) At the time of executing a NEW ON command. (The program is also erased.)
e) When a programs added to, altered or deleted.

For a variable is used for the first time, "0" is assigned if it is a numeric variable and a null string if it is a character variable.
The length of a character string that can be assigned to a character variable is up to 255 characters.

Array variable

Each element of an array variable can be referred to by a subscript.
An array variable is declared by a DIM statement as shown below:

    DIN array variable name (max. value of first dimension
    [max. value of second dimension] .....)

The dimensions of an array variable can be specified for one to a number of dimensions and the maximum value of each subscript can be specified up to 32766. The number of dimensions and the maximum subscript value can be set up to the capacity of memory. If the maximum value of subscripts for each dimension is not greater than "10", the array variable can be used without specifying a DIM statement. Since the value of a subscript starts at "0", the actual number of elements is the sum of the maximum subscript value plus one.
To refer to an array variable, specify the subscript of each dimension by a constant or numeric expression, as shown below.

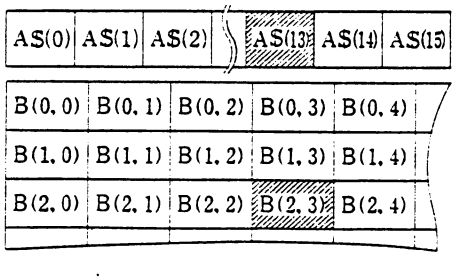
    Array variable name (subscript of first dimension
    [, subscript of second dimension] .....)

(Example 1) Declaration of array

```
DIM A$ (15)
DIM B (12, 15)
DIM C# (6, 9, 20)
```

(Example 2) Referring to array

```
A$ (13)
B (2, 3)
```



1.7 TYPE CONVERSION

BASIC converts the type of a numeric value (constant, variable, calculation result of formula) from one type of a different type. The following rules apply to this conversion.

(1) When a numeric value (constant or variable) of one type is assigned to a numeric variable of a different type, the original type is converted to that of a numeric variable first and its value is assigned. Although assignment of a character string to a numeric variable or a numeric value to a character variable, can be done directly by using a function like ⌜STR$⌟ (which converts a numeric value to a character string) or ⌜VAL⌟ (which converts a character string to a numeric value).

- When a numeric value of the single or double precision type is assigned to an integer type variable, the resulting value is rounded up (fractions of .5 and over are counted up and smaller fractions are disregarded). When the converted value is not in the range from –32768 to 32768, an overflow error results.

(Example)
```
B% = 12.56: PRINT B%
13
```

- When a numeric value of the double-precision type is converted to the single-precision type, rounding up of fractions of .5 and over and disregarding the rest is applied to the seventh digit from the top, and the value thus obtained is assigned.

(Example)
```
C = 3.141592653589793: PRINT C
 3.14159
```

- When a numeric value of the single-precision type is converted to the double-precision type, the effective numeral is a result of rounding up the seventh digit from the top of the converted double-precision type value. The relative error that results from this conversion is as shown below:

Relative error = $|(A\# - B)/B| < 6.3 \times 10^{-8}$

A#: Double precision after conversion
B : Single precision before conversion

(Example)
```
10 A=2.04
20 B#=A
30 PRINT A,B#
40 PRINT 2.04#-B#
RUN
 2.04          2.039999961853027
 3.814697263626599D-08
```

■ Therefore, the error can be made smaller when assigning a constant of up to seven digits to a variable of the double-precision type by writing 「#」 after the numeral and using it as a constant of the double-precision type.

(Example)
```
10 A#=2.04#
20 PRINT A#
RUN
 2.04
```

(2) In the arithmetic calculation of an expression, the precision is matched to the highest one among the numeric values (constant or variable) that appear in the expression. When a calculation result on the right side is assigned to the left side by a LET statement, the assignment is executed following the rule of converting the type of numeric value on the right side to the numeric value on the left side.

(Example)
```
10 D#=6#/7#
20 E#=6/7
30 PRINT D#,E#
RUN
 .8571428571428571       .8571428656578064
```

(3) Logical operations are executed first by applying the method of rounding up fractions of .5 and over and disregarding the rest, thereby making an integer if the numeric value has fractions. If the converted integer is not in the range from –32768 to 32767, an error results.

(Example)
```
PRINT 7.35 OR 16
 23
```

1.8 EXPRESSIONS AND OPERATIONS

An expression can consist of a numeric constant, Hollerith constant, numeric variable, character variable, numeric function, character function and operators that connect them; it's value is either a numeric value or character string. An expression having the value of a character string is specifically called a character expression.

(Example)
```
B+2*A (1)-C+1
X<Y  OR  Z<Y-1
A$+" APPLE "<B$
C$+" ORANGE "
```

Each constant, variable and function used in an expression is called a term. All values in an expression that results in a numeric value must be numeric values and all values of an character expression must be character strings. In other words, numeric values and character strings cannot be operated in the mixed state. However, the form of each function argument can be in the specification of the related function and the form does not necessarily match the form of the whole expression. If, however, an argument happens to be an expression, the same rule is applied to the terms in the expression.

(Examples of incorrect expressions)

A+C$+1	Numeric values 「A」 and 「1」 and a character string 「C$」 are connected by 「+」.
B$+VAL(C$)	Character string 「B$」 and numeric value 「VAL(C$)」 are connected by 「+」.
D$+LEFT$ (E$+5, 4)	Character string 「E$」 and numeric value 「5」 are connected by 「+」.

There are the following five types in the operation in BASIC:

Arithmetic operation, Relational operation, Logical operation,
Character string operation, Function

Arithmetic operation

The arithmetic operators that BASIC can use are listed below in their order of priority.

Order of priority	Operator	Operation contents
①	∧	Exponentiation
②	+, −	Sign (determines whether a term is positive or negative)
③	*, /	Multiplication and division of real numbers
④	¥	Division of integers
⑤	MOD	Remainder
⑥	+, −	Addition and subtraction

When two or more operators of the same priority are used, the operation is conducted from the left to the right. This rule applies to other operators.
Use parentheses to change the order of operation. Also, using parentheses is helpful when the order of operation is not easily understandable.
The following shows the correspondence between ordinary numeric expressions and those in BASIC.

Ordinary algebraic expression	Expression in BASIC
$2X+Y$	2＊X+Y
$X\times Y\div 2$	X＊Y/2
X^2+2X	X∧2+2＊X
$(X^2)^Y$	(X∧2)∧Y or X∧2∧Y
X^{Y^2}	X∧(Y∧2)

An Yen sign is used for the division of integers. If the divisor and multiplicand are real numbers, in the result fractions of .5 or over are rounded up and the rest is disregarded, thereby converting them to integers before the division. Fractions of the quotient are disregarded.

(Example) A = 26.68¥6.99 : PRINT A (27 ÷ 7 = 3 ... with a remainder of 6)

```
3
```

The remainder is calculated by MOD. The MOD calculation leaves the remainder of a division as an integer. When the numeric values are real numbers, the fractions of .5 or over are rounded up and the rest is disregarded before the calculation.

(Example) PRINT 26.68MOD6.99 (27 ÷ 7 = 3 ... with a remainder of 6)

```
6
```

When division is performed using "0" as the divisor or when overflow occurs, an error results.

Relational operation

Relational operators are used to compare two numeric values or characters, and relational operations have a high priority next to arithmetic operations among all operations using operators. The operation value is expressed as "−1" if the comparison result is true and "0" if not true.
Character strings are compared by comparing the character code of each character from left to right. When all character codes are equal, the two character strings are equal in value. If there are different character codes, the character string having a higher character code is determined to be larger. When the character string of one of the two strings being compared ends in the middle of comparison, the character string which still has characters is determined to be larger. A blank in a character string is considered to be character code &H20.

The comparison result can be used to change the flow of a program using an IF statement. The following shows the types of relational operators. They all have the same priority order.

Operator	Meaning	Example	
		Comparison expression in BASIC	Numeric expression
=	Equal to	X = Y	X = Y
<>, ><	Not equal to	X<>Y or X><Y	X ≠ Y
<	Smaller than	X<Y	X<Y
>	Greater than	X>Y	X>Y
<=, =<	Equal to or smaller than	X<=Y or X=<Y	X≦Y
>=, =>	Equal to or greater than	X>=Y or X=>Y	X≧Y

```
(Example)    PRINT 12>11, 12=11
              -1          0
```

Logical operations

In a logical operation, first, the numeric value is converted to the complement of "2" in 16 bits (to an integer in the range from –32768 to 32767), and then each bit is inverted or two numeric values are processed for each bit. The following shows each logical operation and operation result of each bit in the order of operation having a higher priority order. (X and Y are the values of two numeric values in the same bit position.)

NOT (Negation)

X	NOT X
1	0
0	1

NOT X

This means "It is not X". If X is true (–1), the result is not-true (0). If X is not-true (0), the result is true (–1).

AND (Logical product)

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

X AND Y

This means "it is X and Y at the same time". The result is true (–1) only when both of X and Y are true (–1).

OR (Logical sum)

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

X OR Y

This means "it is X or Y". The result is true (–1) when at least one of X or Y is true (–1). That is, if X and Y are both not-true (0), the result is not-true (0).

XOR (eX clusive OR)

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

X XOR Y
This means "X and Y are not the same in the value". When only one of X and Y is true (−1), the result is true (−1).

IMP (IMPlication)

X	Y	X IMP Y
1	1	1
1	0	0
0	1	1
0	0	1

X IMP Y
This means "X is included in Y". Only when X is true (−1) and Y is not-true (0), the result is not-true (0).

EQV (EQuiValence)

X	Y	X EQV Y
1	1	1
1	0	0
0	1	0
0	0	1

X EQV Y
This means "X and Y are the same in the value". When both of X and Y are true (−1) or not-true (0), the result is true (−1).

The following shows operation examples using actual numeric values. The contents in ($)_2$ are the binary number (16-bit complement of "2") delimited for 4-bit groups.
(Example)

- −1 OR 0 = −1
  −1 = $(1111\ 1111\ 1111\ 1111)_2$
  0 = $(0000\ 0000\ 0000\ 0000)_2$
  therefore, −1 OR 0 = −1
- 23 AND 7 = 7
  23 = $(10111)_2$
  7 = $(00111)_2$
  therefore, 23 AND 7 = 7
- 16 XOR 6 = 22
  16 = $(10000)_2$
  6 = $(00110)_2$
  therefore, 16 XOR 6 = $(10110)_2$ = 22

- 12 IMP 5 = −9

  12 = $(0000\ 0000\ 0000\ 1100)_2$

  5 = $(0000\ 0000\ 0000\ 0101)_2$

  therefore, 12 IMP 5 = $(1111\ 1111\ 1111\ 0111)_2$ = −9

- 16 EQV 6 = −23

  16 = $(0000\ 0000\ 0001\ 0000)_2$

  6 = $(0000\ 0000\ 0000\ 0110)_2$

  therefore, 16 EQV 6 = $(1111\ 1111\ 1110\ 1001)_2$ = −23

  The relationship X EQV Y = NOT (X XOR Y) is possible. Also, there is the relationship NOT X = − (X + 1).

As done by relational operators, logical operators can change the flow of a program using an IF statement. In this case a logical operator can be connected with two or more relational operators.

(Example) IF A = 0 OR B = 0 THEN 500

Execution branches to Line No. 500 when A is 0 or B is 0.

Character string operation

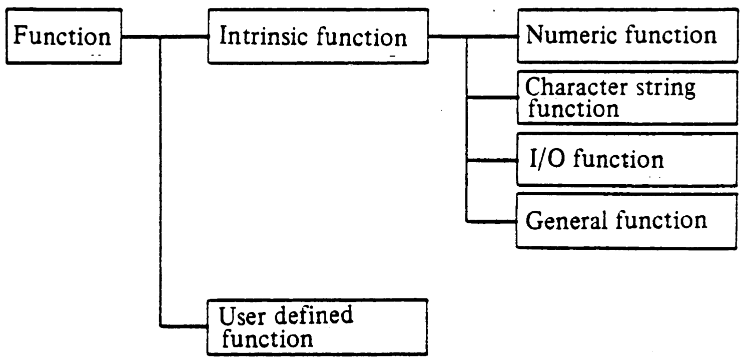
Character strings can be connected using operator 「+」. This operation is very convenient to make a new character string collecting partial character strings or to slightly change part of a character string.

(Example)

```
10 A$= " Hitachi " : B$= " MB−S1 " : C$= " BASIC "
20 D$=A$+B$+C$
30 PRINT D$
RUN
Hitachi MB−S1 BASIC

Ready
```

Functions

The functions of BASIC have been expanded from the handling of arithmetic numeric values to being able to handle character strings, and it gives a numeric value or character string to a specified argument following a constant rule. BASIC is provided with many types of functions as shown below.



An intrinsic function is a function possessed by BASIC and one or more arguments enclosed in parentheses follow the function name. The number of arguments and whether each argument has a numeric value or character string depends on the function. Some functions do not have an argument.

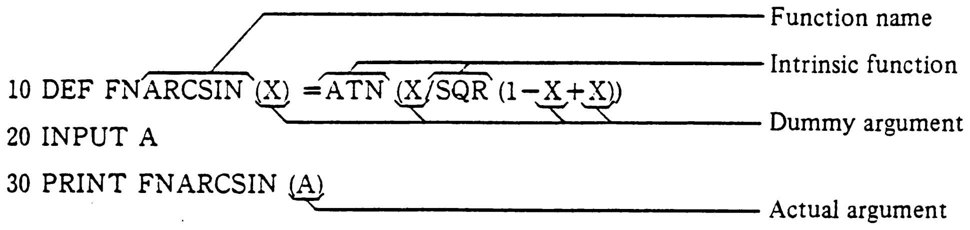
The argument can take a form of an expression. Also, a function can be freely used in an expression, in the same way that constants and variables can be used.

(Example)

```
ATN (X/SQR (1-X * X))
```

In addition to prepared functions, BASIC has user defined functions. With this type function, the number of arguments, the argument attributes (numeric value or character string), and what value (numeric value or character string) should be given to each argument can be defined in the program, and function names starting with 「FN」 are used. The definition of each function is done by a DEF FN statement and the definition expression is written after the function name that starts with 「FN」. The definition expression must be a single expression. As an example, below is a numeric arc tangent.

(Example)



```
10 DEF FNARCSIN (X) =ATN (X/SQR (1-X+X))
20 INPUT A
30 PRINT FNARCSIN (A)
```

Priority order of operation

The operation is conducted in the following order:

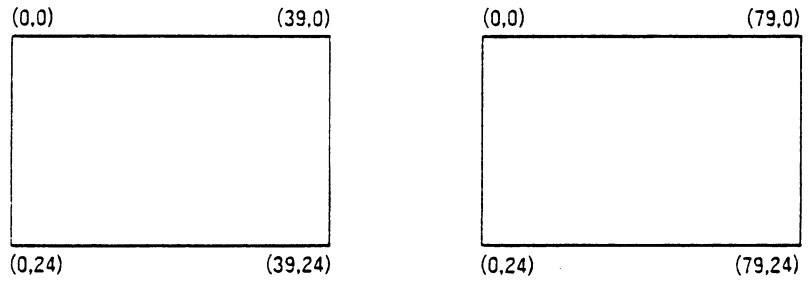
① Part enclosed in parentheses
② Function
③ Exponent (Exponentiation)
④ Minus sign (−)
⑤ $*$, $/$
⑥ ¥
⑦ MOD
⑧ +, −
⑨ Relational operators (<,>, =)
⑩ NOT
⑪ AND
⑫ OR
⑬ XOR
⑭ IMP
⑮ EQV

1.9 COORDINATE SYSTEMS

BASIC uses many coordinate systems to display characters and graphics on the screen.

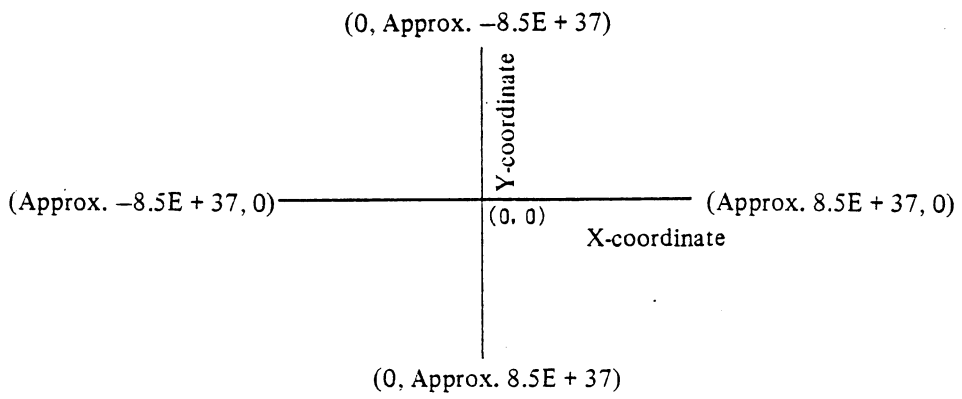
Coordinate systems in S1 BASIC

The coordinate systems in S1 BASIC can be divided into two groups, a character coordinate system and a graphic coordinate system.
The character coordinate system is used to display characters on the screen, and there are two types of 40 x 25 (width x height) and 80 x 25, depending on the number of digits (40 digits or 80 digits which can be specified by a WIDTH statement) that are to be displayed on each line. The coordinate system is specified by a LOCATE statement.



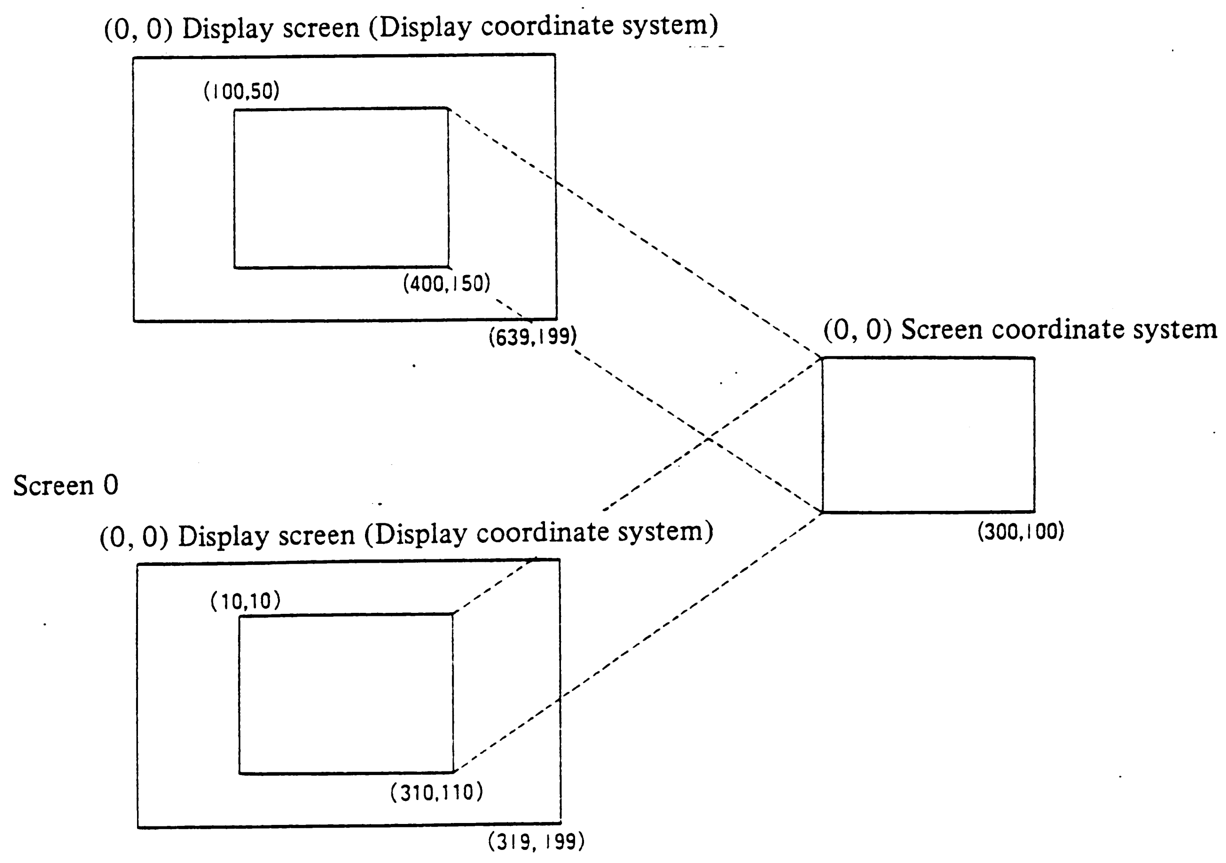
The graphic coordinate system has two coordinates, one is the world coordinate system which can handle a virtual (logical) coordinate system and the other is a screen coordinate system using the display screen. The display screen is available in two types, 640 dots x 200 dots (width x height) and 320 dots x 200 dots by specifying the selection with a SCREEN statement.

The world coordinate system is a large coordinate system handled by the user. Its size ranges from approximately –8.5E + 37 to approximately 8.5E + 37 for both of the width and height, and cannot exceed this range.



The screen coordinate systems exist on the display screen and each dot on the screen corresponds to each point of the coordinates. The reference point (0, 0) is at the upper left corner and any point on the display screen can be specified. When the whole display screen is used in the screen coordinate system, it is specifically called the display coordinate system.

Screen 1 or the initial state


(0, 0) Display screen (Display coordinate system)
(100,50)
(400,150)
(639,199)
(0, 0) Screen coordinate system
(300,100)
Screen 0
(0, 0) Display screen (Display coordinate system)
(10,10)
(310,110)
(319, 199)

To specify graphic coordinates, input an instruction for the world coordinates or screen coordinates. On instructions that are used to specify the world coordinates, if omission of a coordinate specification is permitted, a coordinate called the "last reference point" (hereinafter referred to as LP) can be used.

LP means the point that was specified at the end of the last operation, and it has the value of the coordinate that was last specified in a graphic instruction using the world coordinates.

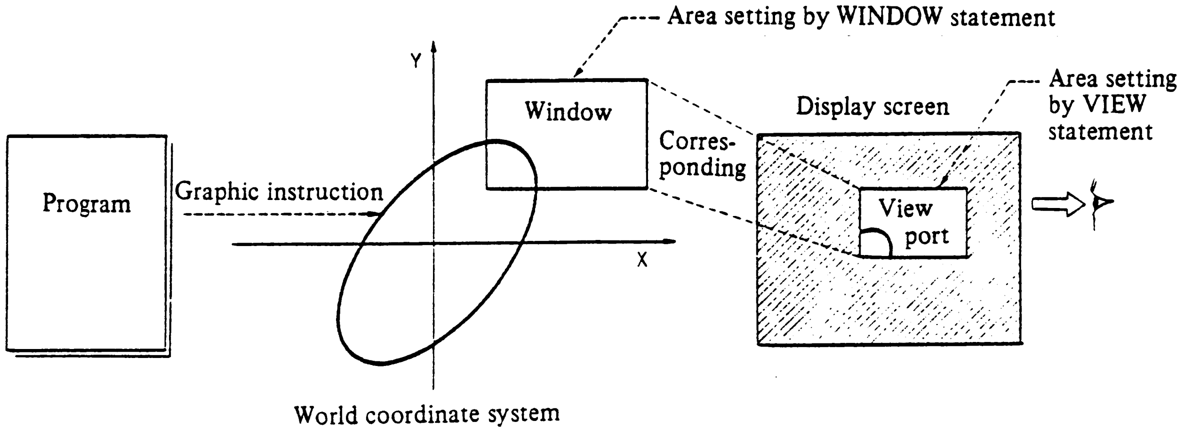
(Example)

10 CONNECT (10, 10) – (50, 50) ..... (50,.50) is the LP.

20 LINE – (100, 100), PSET, 3, B ..... Draws a box using (50, 50) and (100, 100) as the diagonal points.

1.10 WINDOWS AND VIEW PORTS

The world coordinates explained in the preceding section are basically a virtual coordinate system, and the coordinates that can be actually displayed and observed are the screen coordinates. Therefore, the contents of world coordinates can be observed by making a specific area of the world coordinates the screen coordinates. This specific area is called a "window" and it is specified by a WINDOW statement in S1 BASIC.
When a window is specified in world coordinates, the next specification needed is the location of the display of the window contents on the display screen. This specification is given by a VIEW statement in S1 BASIC, and the specified area is called a "view port". The screen coordinates are the coordinates of this view port.



Set the window and view port areas as shown below.

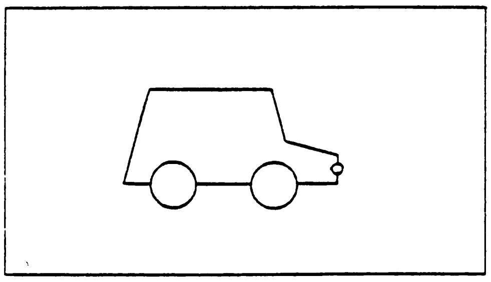
State / Area	Initial state	Execution of SCREEN statement (fineness)		Execution of WINDOW statement	Execution of VIEW statement
		0 (Note)	1		
Window	(0, 0) – (639, 199)	(0, 0) – (639, 199)	(0, 0) – (639, 199)	Specified area	–
View port	(0, 0) – (639, 199)	(0, 0) – (319, 199)	(0, 0) – (639, 199)	–	Specified area

(Note) When SCREEN 0 is executed, the window and view port do not correspond 1:1 in their lateral direction. Specify (0, 0) – (319, 199) for the area in the WORLD statement to have the two correspond exactly.

Movement, reduction and enlargement of graphics

Graphics can be moved, reduced or enlarged without changing the graphic instruction by operating using the window and view port. The following explains these changes using a simple example.
First, draw a starting sketch (a car for example).

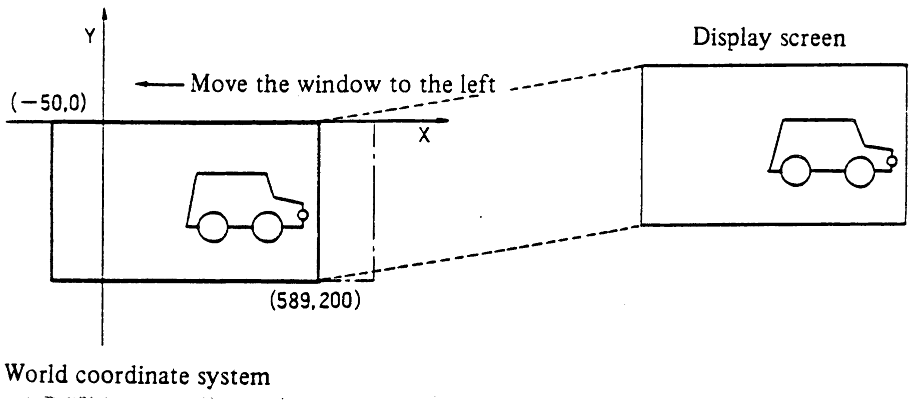
```
10  SCREEN 1
20  GOSUB 40
30  END
40  'CAR
50  CONNECT (100, 130)-(130, 70)-(310, 70)-(360, 100)-(450, 110)-(450, 130)-(100,
    130)······Body
60  CIRCLE (170,130),35,3······ Rear wheel
65  PAINT  (170,130),0,3
70  CIRCLE (350,130),35,3······ Front wheel
75  PAINT  (350,130),0,3
80  CIRCLE (450,120),10,4······ Headlight
85  PAINT  (450,120),0,4
90  RETURN
RUN
```



(1) Movement of graphics

To move the car slightly, the car seems to have moved towards right by moving the window toward the left.

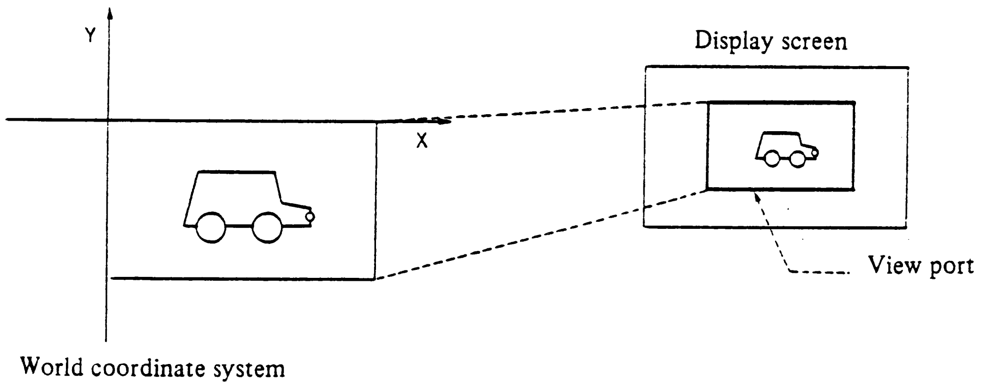
```
100  SCREEN 1
110  WINDOW (-50, 0)-(589,200)
120  GOSUB 50
130  END

RUN 100
```



(2) **Reduction of graphics**

Now to reduce the size of the car. The car looks smaller if the view port is made smaller.

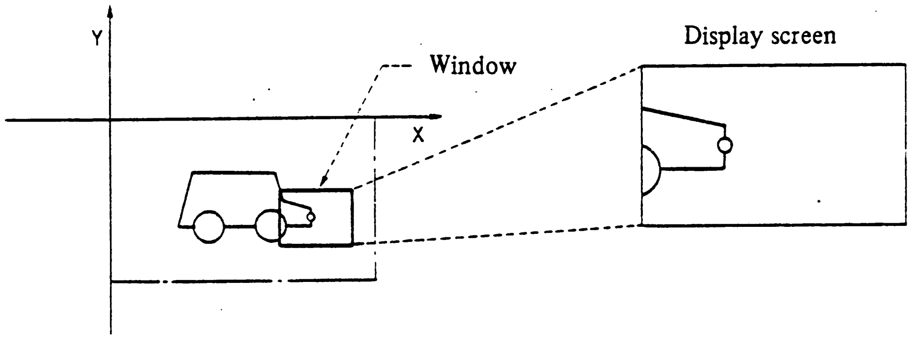
```
140  SCREEN 1
150  VIEW (150, 50)−(450, 150)
160  GOSUB 50
170  END

RUN 140
```



(3) Enlargement of graphics

Now to enlarge the front of the car. The front looks larger by specifying this part in the WINDOW statement.

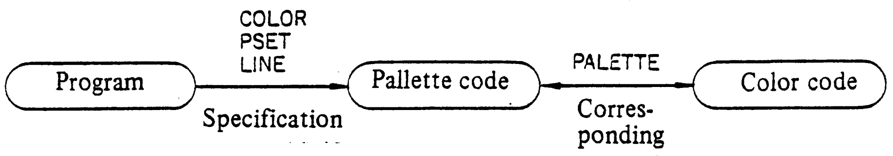
```
180  SCREEN 1
190  WINDOW (350, 100)-(550, 150)
200  GOSUB 50
210  END

RUN 180
```



World coordinate system

1.11 COLOR DISPLAY

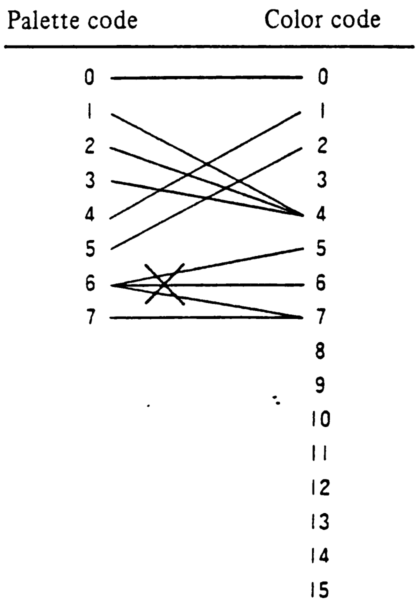
In the case of S1 BASIC, eight colors among 15 usable colors can be freely selected by specifying them using a "Palette'.
Each palette is given a Code No. in the range "0" to "7", and the color can be quickly changed by assigning a color out of the 16 colors usable in each palette. In a program, colors are specified as palette codes, and correspondence between each palette code and color code is realized by PALETTE statements.



The initial state of assigning palette codes and color codes is as shown below.

Palette code	Color code	Color
	0	Black
	1	Dark blue
	2	Dark red
	3	Dark magenta (purple)
	4	Dark green
	5	Dark cyan (aquamarine)
	6	Dark yellow
	7	White
0	8	Black
1	9	Light blue
2	10	Light red
3	11	Light magenta (light purple)
4	12	Light green
5	13	Light cyan (light aquamarine)
6	14	Light yellow
7	15	Bright white

One color code may be assigned to two or more palette codes, but two or more color codes cannot be assigned to one palette code.



In this case, palette codes 1, 2 and 3 are all dark green. Palette code 6 cannot be assigned to color codes 5, 6 and 7.

(Example) PALETTE 1, 4 . . . . . This assigns dark green (Color code 4) to Palette code 1.

1.12 INTERRUPTS

The core of a computer is called the CPU (Central Processing Unit). Althought the CPU can faithfully execute a series of instructions given one after another, it cannot process other things. Therefore, to have a computer process one item with high priority while the CPU is processing another item, the request must be notified to the CPU. The action of this notification is called an "interrupt" and when an interrupt is applied, the CPU temporarily stops the current processing and executes the specified interrupt processing.
Since the MB-S1 can be connected with various peripheral devices, BASIC provides functions to handle interrupts from these devices.
The table below shows the types of interrupts that are supported by S1 BASIC.

O: Usable x: Unusable

Interrupt type	Statement to refer to	S1 BASIC
Interrupt from communication port	ON COM (n) GOSUB	O
Interrupt when an error occurs	ON ERROR GOTO	O
Interrupt from interval timer	ON INTERVAL GOSUB	O
Function key interrupt	ON KEY (n) GOSUB	O
Mouse interrupt	ON MTRIG (n) GOSUB	O
Light pen interrupt	ON PEN GOSUB	x
Joystick interrupt	ON STRIG (n) GOSUB	O

1.13 FILE DESCRIPTORS

BASIC handles information of all I/O devices under the unified concept of "files". The term "file" means a group of information having a meaning. A file descriptor distinguishes the information for the related device and information name.

Configuration of file descriptor

A file descriptor consists of the following character strings:

" [<device name> :] [(<option>)] [<file name>] "

Normally a file descriptor is expressed as a Hollerith constant enclosed by two double quotation marks 「"」, but it can also be expressed by a character variable or character expression.

Any of the <device name>, <option> and <file name> can be omitted.

When specifying the <device name>, a colon 「:」 must be written immediately after it even when the <file name> is not specified. The reason for this is that, if the colon is not specified, the file descriptor may be regarded as a file descriptor in which the device name is omitted and which consists of only a file name.

(1) **Device name**

A device name is given for each I/O device, and it shows the location of the file specified by the file descriptor. In the case of BASIC, the device names are defined for the I/O devices shown in the table on the next page.

When BASIC is expanded in the future, more device names will be defined.

BASIC assumes 「CAS0:」 when the device name specification is omitted.

No.	Part name		Device name	Input	Output	Remarks
1	Keyboard		KYBD:	O	x	
2	Screen		SCRN:	x	O	
3	Printer	0	LPT0:	x	O	
4		1	LPT1:	x	O	Additional printer card MP-1810 is necessary.
5		2	LPT2:	x	O	"
6	RS-232C port	0	COM0:	O	O	Connector option MP-9732 is necessary.
7		1	COM1:	O	O	Additional RS-232C card MP-1820 is necessary.
8		2	COM2:	O	O	"
9		3	COM3:	O	O	"
10		4	COM4:	O	O	"
11	Cassette		CAS0:	O	O	Specification of device name may be omitted.

(2) **Option**

The term "option" means giving a special specification to an input/output device, and input/output with a device which permits "optional specification", the specification can be made using alphanumerics consisting of up to four characters and enclosed in parentheses ⌈(　)⌋. In BASIC, options are not permitted for equipment other than the control of RS-232C lines.

(3) **File name**

A file name is used to distinguish a file from many other files on the specified input/output device. Specify the <file name> using up to eight characters (characters or signs). Use of a colon ⌈:⌋, parentheses ⌈(⌋ and ⌈)⌋, and any character which has a character code "0" (&H00) or "255" (&HFF) cannot be used.

The <file name> is not needed by file descriptors other than Cassette (CAS0:).

Examples of file descriptor:

" CAS0 : PROG1 "	PROG1 file on a cassette tape
" PROG2 "	PROG2 file on a cassette tape
" SCRN : "	Screen
" LPT0 : "	Line printer 0
" COM0 : (S8N1) "	RS-232C Port 0

1.14 ERROR MESSAGES

When an error occurs during program execution, BASIC temporarily stops execution, displays an error message and then displays 「Ready」, and accepts a command. Variable values and the program remain as they are at the time of a temporary stop.
An error made in a program is not detected until the line where the error is made is executed.
An error message corresponding to a statement in a program is displayed with the Line No. where the error occuoused, in the format shown below.

<error message> In <Line No.>

If there is an error in a command or in a statement that is executed as a command, an error message only is displayed. For error codes and the meanings of error messages in BASIC, refer to APPENDIX 5 ERROR MESSAGES.
Also, BASIC is provided with error processing instructions and functions so that program execution can be continued even when an error has occurred.

1.15 INTRODUCTION TO NEW FUNCTIONS

The following provides easy explanations of music, image generator, mouse and KANJI processing, among the new functions in S1 BASIC.

Music

S1 BASIC can provide music with PLAY statements and a rich variety of sounds with SOUND statements.

(1) **PLAY statement**

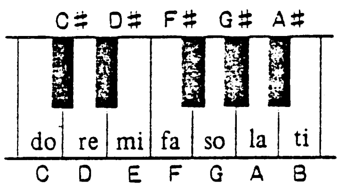
To have the computer play music, input a PLAY statement followed by a character string called "music data". Give the following instruction:

PLAY "CDEFGAB"

- The sounds of "do", "re", "mi" ... can be specified as the music data using alphabetic characters as shown by the above instruction. The example specifies white notes only, but black notes can also be produced, as shown in the following example:

  PLAY "CC # DD # EFF # GG # AA # B"

  In addition to the method shown above, black notes can be specified by one of the following three methods.



Scale	Specification method	
	White note	Black note
do	C	C♯, C+, Db, D−
re	D	D♯, D+, Eb, E−
mi	E	——
fa	F	F♯, F+, Gb, G−
so	G	G♯, G+, Ab, A−
la	A	A♯, A+, Bb, B−
ti	B	——

- Sounds of up to eight octaves can be set by Command「O」and a numeric after the「O」.

  PLAY "CDEFGABO5C"

  Thus, specify「O5」to obtain higher "do". The computer is initially set to「O4」.

- There is another method to set a scale. It uses「N」and the object sound is produced by a character string.

  PLAY "O4CN37"

  The first sound (O4C) and the next sound (N37) are the same. By changing the numeric value in this way, a series of sounds from "do" (C) of Octave 1 to "ti" (B) of Octave 8 can be produced. One change of the numeric value changes the tone in semi-tone units, and these changes are summarized in the table below.

Correspondence between intervals and special intervals

Interval	Octave	1	2	3	4	5	6	7	8
do	C	1	13	25	37	49	61	73	85
do ♯	C ♯	2	14	26	38	50	62	74	86
re	D	3	15	27	39	51	63	75	87
re ♯	D ♯	4	16	28	40	52	64	76	88
mi	E	5	17	29	41	53	65	77	89
fa	F	6	18	30	42	54	66	78	90
fa ♯	F ♯	7	19	31	43	55	67	79	91
so	G	8	20	32	44	56	68	80	92
so ♯	G ♯	9	21	33	45	57	69	81	93
la	A	10	22	34	46	58	70	82	94
la ♯	A ♯	11	23	35	47	59	71	83	95
ti	B	12	24	36	48	60	72	84	96

- The length of sound can be changed by command character 「L」. Supposing that the tone of "do" in the above example has a length of a quarter note, it can be changed to a length of an eighth note by the following instruction:

  PLAY "CL8C" .... "L8" shows an eighth note.

  The length of the second "do" should now be half of the length of the first "do". In the initial state (default), the length of each sound is set to "4". The greater the value of L, the shorter the sound length, and the sound length is variable in the range from "1" to "64".
- There is another method to change the sound length, and that is, describe a period next to the sound name. For example, the instruction shown before can be rewritten as follows:

  PLAY "L4C."

  One or two periods may be used.
- The command character to show a rest is 「R」. Write a numeric showing the length after 「R」. The numeric following R is in the range from "1" to "64", the same as the numerics for L.

```
10 PLAY "L4CDER4"
20 GOTO 10
```

  A period can be used after a rest.
- The basic points of how to produce tones have been explained, and now music data of a simple tune is shown below. Execute the following program:

```
10 PLAY "O4L8CCGGAAL4GL8FFEEDDL4C"
```

  Upon hearing this, you may wish to quicken the tempo or make it slower. Use Command T in such a case. By specifying a numeric that shows the speed after T, the tempo is changed. Add the following line to the preceding program:

```
5 PLAY "T150"
```

  You see that the tempo is slightly quickened. The default value of tempo is 120, and T can have values in the range from "32" to "255". This value can be determined by the number of quarter notes counted per minute. Specify the tempo command before the music data.

■ The explanation that has been given so far is on monophonic (single tone) playing, but polyphonic (chord) playing is also possible.

```
PLAY "C", "E", "G"
```

You now hear a chord consisting of "do", "mi" and "so". A chord of up to six notes can be produced by delimiting each character string in the music data with a comma (an optional card is needed to produce chords of 4 to 6 notes). Since the tempo and note length can be specified individually, even a tune having various parts can be played by the computer.

The command used to change the sound volume of each part is 「V」. The default value of V is 8, and it can take a value in the range from "0" to "15".

```
PLAY "C", "V15E", "V1G"
```

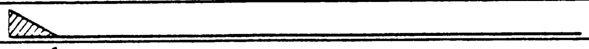
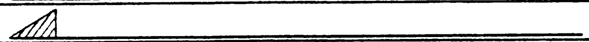
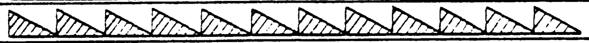
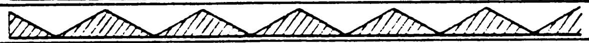
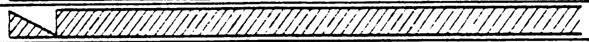
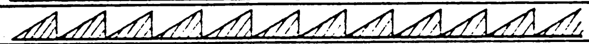
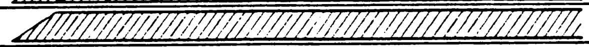
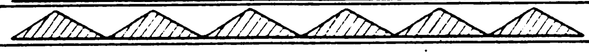
The "mi" sound should be louder and the "so" sound should be softer now.

■ Envelope

The term envelope refers to the sound waveform. Since there are various sounds, there are various envelope patterns.

16 patterns (actually 8 patterns) can be selected by a PLAY statement in S1 BASIC. The character used to specify the pattern is 「S」 and S takes a value in the range from "0" to "15". Envelope patterns 0 ~ 15 are shown in the table below.

Correspondence between envelope patterns and S command value (n)

n	Envelope pattern (Vertical axis: volume, Horizontal axis: time)
0 −3.9	
4 −7.15	
8	
10	
11	
12	
13	
14	

Command character 「M」 must be used in a set with 「S」 to play a tune using the envelope.

The M command determines the frequency of the envelope and it takes a value in the range from "0" to "65535". This value can be determined from the following formula:

$$M = \frac{\text{Cycle (seconds)} \times 1.008\ \text{(MHz)}}{256}$$

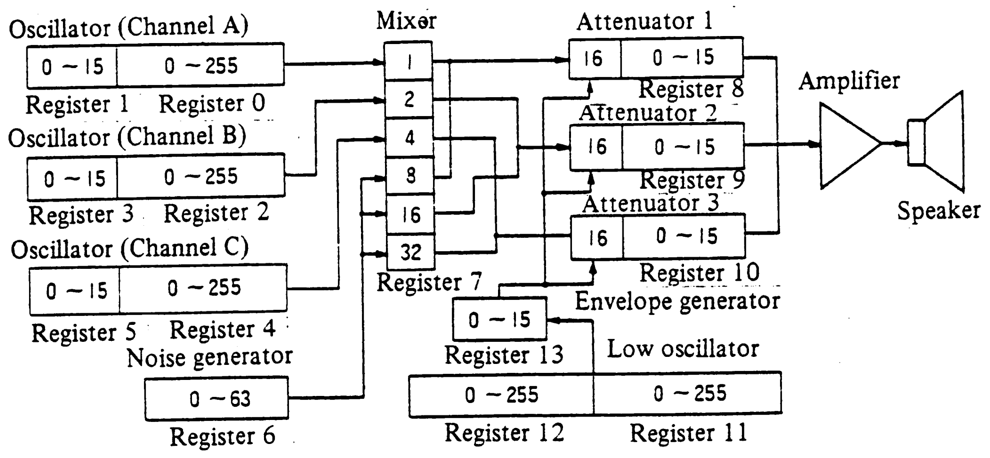
Try various S and M values and listen to the difference created by different envelope patterns.

There are various other functions; for further details, refer to the PLAY statement section.

(2) SOUND statement

The preceding section explained how to play music. However, the PLAY statement cannot produce the sounds of UFOs or laser beams. This section describes how to produce sounds using a SOUND statement.

The sound generator used in the S1 consists of an oscillator block, a mixer block, an attenuator block, a noise generation block, an envelope generation block and a mixing amplifier block, as shown below.



Sound generator block diagram

The number of registers provided for these blocks is 14, and the function of each register is shown in the table below.

Register No.	Data 7	6	5	4	3	2	1	0	Operation	Remarks
0	Lower 8-bit data								Channel A Scale	The scale is expressed by 12-bit data.
1					Higher 4-bit data					
2	Lower 8-bit data								Channel B Scale	
3					Higher 4-bit data					
4	Lower 8-bit data								Channel C Scale	
5					Higher 4-bit data					
6	5-bit data								Noise frequency	
7			Noise			Tone			Selection of noise tone channel	
			C	B	A	C	B	A		
8				M	4-bit data				Channel A Volume	When M=0, lower bits adjust the volume. When M=1, the envelope is effective.
9				M	4-bit data				Channel B Volume	
10				M	4-bit data				Channel C Volume	
11	Lower 8-bit data								Envelope period	
12	Higher 8-bit data									
13					Refer to SOUND statement				Envelope waveform	

The following describes the details of the register functions.

- Registers 0 and 1 are contained in the oscillator as a pair, and four bits of Register 1 and eight bits of Register 0 determine the frequency of oscillation. The value of data to be written can be obtained as follows:

$$\frac{1.008\ (\text{MHz})}{16 \times \text{frequency}} = \text{Data value}$$

  This pair of Registers 0 and 1 is called Channel A. Registers 2 and 3 and Registers 4 and 5 are also in pairs, each forming an oscillator, and these are called Channels B and C, respectively. Therefore, the sound generator in the S1 can produce three chords.
- Register 6 functions as a noise generator. This register also designates a frequency in the same way that Channel A does. However, only five bits of data are effective and the higher 3 bits of data are ignored.
- Register 7 functions as a mixer. The tone and noise from the three oscillators are mixed and the results are sent to the attenuator. A switch for the tone and noise is provided for each bit.
- Registers 8 through 10 are connectored to attenuators. The lower four bits designate the attenuation, that is, the volume. The value taken by these bits is in the range from "0" to "15". If a value of "16" is taken, the next bit is set and it selects the envelope explained later. An oscillator and attenuator make a pair and form a channel.
- Registers 11 and 12 designate the envelope frequency. The value of data to be written is determined as follows. Register 11 is the lower order and Register 12 is the higher order.

$$\frac{1.008\ (\text{MHz})}{256 \times \text{frequency}} = \text{Data value}$$

- Register 13 determines the waveform of the envelope. The value taken is in the range from "0" to "15", and there are eight waveform types.

Since the sound generator cannot produce sound when music data is specified as done in a PLAY statement, a program must be prepared in which the written data is varied effectively. Therefore, a SOUND statement is used in a set with a FOR statement in most cases. For example, prepare a program as shown below and execute it.

(Example) Sound of a gun firing

```
10 SOUND 6, 12
20 SOUND 7, 247
30 SOUND 8, 16
40 SOUND 11, 96
50 SOUND 12,9
60 SOUND 13, 9
70 SOUND 13,8
```

Sound of waves

```
10 SOUND 7, 7
20 FOR I=0 TO 15
30 SOUND 8, I
40 FOR J=1 TO 100 : NEXT J
50 NEXT I
60 FOR I=15 TO 0 STEP-1
70 SOUND 8, I
80 FOR J=1 TO 100 : NEXT J
90 NEXT I
```

Image Generator (IG)

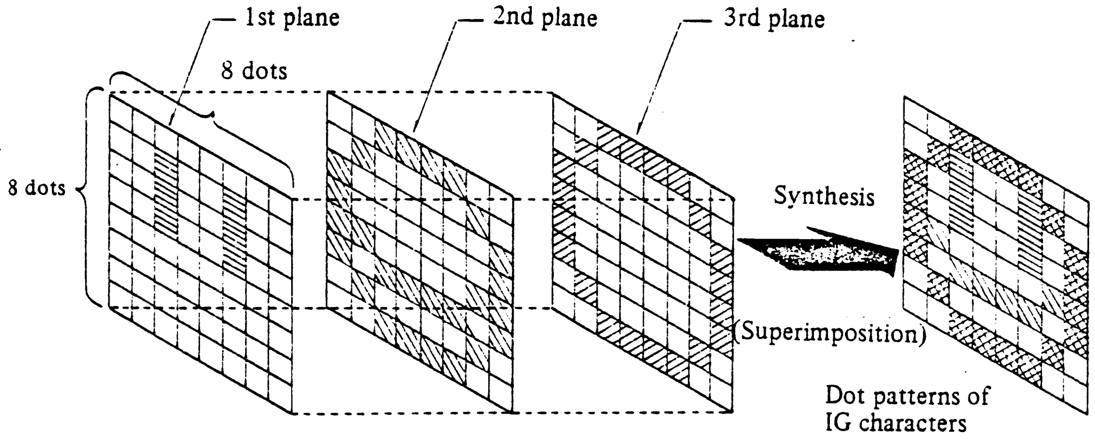
Patterns of characters that are output to the screen or printer are written to the image generator in advance, and the image generator can then output the character pattern to the screen or printer when an instruction is input from the keyboard or instructed by a BASIC program.

The device into which patterns are written under a set rule is called a character generator, and it cannot display patterns other than those already written to it. In other words, the character generator cannot change the character patterns. This is because the character generator is equipped with a ROM (Read Only Memory) but no memory to which data can be written.

This computer has an image generator (IG) in addition to a character generator, allowing character patterns to be changed without changing the ROM.

A RAM (Random Access Memory) to which data can be written and erased, is used for the IG and any patterns can be written to it.

With the IG, 256 character patterns can be freely rewritten. The size of one character pattern that can be created by the IG is the same as the size of one character shown in the character code table and it consists of eight dots each for the height and width. A square resulting from overlaying horizontal eight dots in eight layers is the size of one character pattern. The IG can control all 64 dots in one character in the unit of dots and furthermore, it can color all dots in up to eight different colors.
Since the RAM used in the IG is allocated into three planes (an area consisting of 8 x 8 dots), the Palette No., which is a color, can be determined for each dot by combining the patterns of each plane.
Since up to eight colors can be used in one character pattern when using the IG, the colors of characters used in a game can be changed in many ways. Also, since the characters can be displayed by using a PRINT statement in BASIC, the characters can move very quickly.
Thus, with this computer, the color of each dot can be specified in the unit of characters, and all 256 character patterns can be freely changed.



- IG definition method

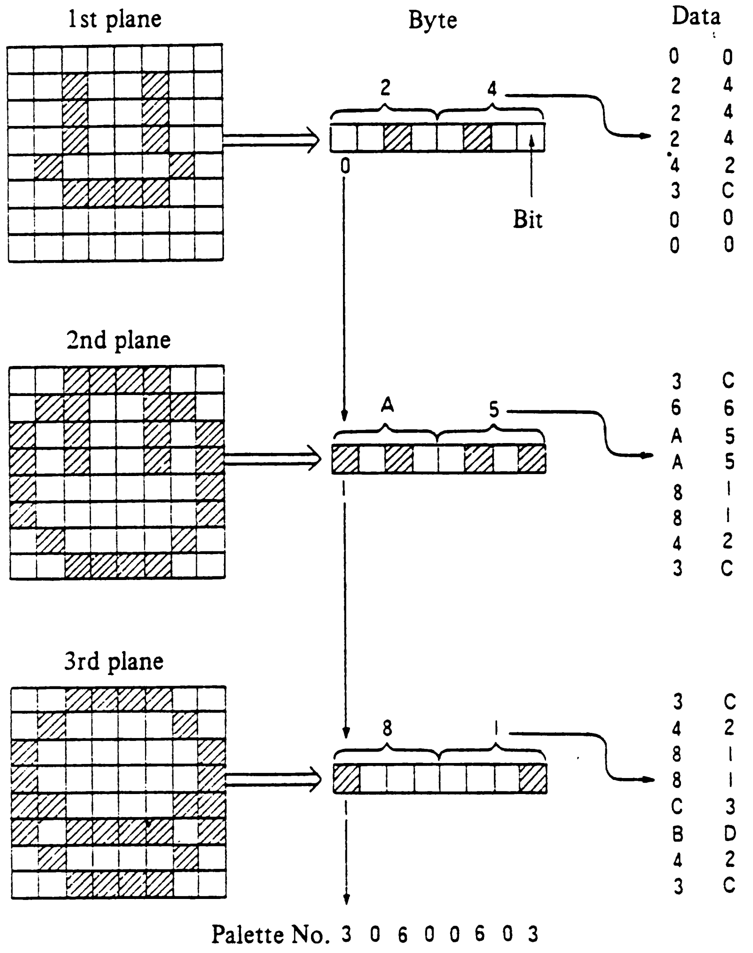
  We have explained that eight colors are available for each dot. We have also explained that one character consists of 8 x 8 dots. Therefore, it is not so easy to define IG characters. However, an IG$ statement is prepared to perform this using a BASIC statement. The method of definition is similar to assigning a character string to an array variable. For example, it is written as follows:

  ```
  IG$ (32) = "00242424423C0000", "3C66A5A581814 23C", "3C428181C3BD423C"
  ```

  The numerals in parentheses are not subscripts but are character codes. The value of a character code is in the range from "0" to "255", which is the same as the range of characters for the character generator.

  The three character strings on the left side, which are delimited by commas, correspond with the data of the first, second and third planes, respectively, and they define the patterns in the corresponding planes.

Each plane consists of 8 x 8 dots, as shown in the diagram below. Among these 64 dots, the eight horizontal dots are allocated to a memory of one byte. Therefore, each plane consists of eight bytes.



Since one byte is expressed by two hexadecimal numbers, the data of one byte is expressed by two characters. Therefore, the data for each plane is expressed by 16 hexadecimal numbers.
The Palette No. is determined by the combination of each bit corresponding to the three planes, and a color corresponding to each Palette No. is displayed.
When starting BASIC, the IG is set to the same pattern that is set in the character generator.
The next explanation concerns how to display the IG contents.
The mode must be switched to the IG mode from the normal character mode, and this can be done easily using a COLOR statement. In the parameters of a COLOR statement, a third parameter can be specified in addition to the character color and background color, and the third parameter is called the IG switch. When the IG switch is set to "1", the IG display mode is set, and when it is cleared ("0"), the normal character mode is set. Input as follows:

```
COLOR , , 1
```

The method to return to the command level is different from the ordinary case. Each character and pattern that were generated by the IG definition can be displayed in this state using a PRINT statement. Try it.
In this way, the display of IG contents can be switched by a COLOR statement. Therefore, normal characters and IG characters can be displayed in the mixed state by using a COLOR statement in a program.

Mouse

The mouse is related to a joystick and light pen and it is an input devices. It is called a mouse since its shape is similar to a mouse.
The mouse has a ball that rotates at its underside. When the mouse is moved, the ball rotates and the movement distance and position of the mouse are known by the rotation of the ball. The ball is not rotated by finger but the mouse moving on a board. Therefore, it is easy to use. Also, since the movement distance and position of the mouse are known exactly, it enables the computer user to input more detailed screen information than using a light pen.

(1) **Movement of the mouse**

The MREAD function gives the information on the mouse's position and movement according to the argument of the function. When the argument is "0", the value of the mouse's X coordinate (lateral direction) is informed to the computer. When it is "1", the value of the mouse's Y coordinate (vertical direction) is informed to the computer.

```
PRINT MREAD (0) ; MREAD (1)
```

This displays the values of the X and Y coordinates. Move the mouse slightly and execute the same instruction again. Different values will be displayed. These values are the same as the values of the screen coordinates, and they are in the range from "0" to "639" for the X coordinate and from "0" to "199" for the Y coordinate.
There is another argument in the MREAD function and it relates with the direction of mouse movement. When the argument is "2", the value of mouse movement in the lateral direction is informed. If it is in the positive (right) direction, "0" is informed and if it is in the negative (left) direction, "−1" is informed. Argument "3" gives the values in the vertical direction in the same way. Execute the following program and move the mouse. You will understand the operation of the MREAD function from it.

```
10  PRINT  MREAD (0) ; MREAD (1) ; MREAD(2); MREAD(3)
20  GOTO  10
```

(2) **Operation of mouse buttons**

The mouse is given two switches and they are called trigger buttons. The MTRIG function informs the state of the trigger buttons. Information related to interrupts can be obtained using the mouse.
The argument is selected depending on which state of the two should be informed. When the argument is "1" or "3", the state of Button A is informed. When the argument is "2" or "4", the state of Button B is informed.

Execute a simple program, as shown below, in order to check the operation of the MTRIG function:

```
10 PRINT MTRIG (1) ;MTRIG (2)
20 GOTO 10
```

(3) **Interrupt caused by mouse**

To allow an interrupt by the mouse, use an MTRIG statement. The following instruction is used to allow an interrupt by Button A of the mouse:

```
MTRIG (1) ON
```

In order to process an interrupt from the mouse, this instruction must always be executed. Execute this instruction first and then press the button. The operation branches to the interrupt processing routine.

To prohibit interrupts from the mouse, use OFF instead of ON. For example, input the following:

```
MTRIG (1) OFF
```

This prohibits interrupt processing of Button A.

There is a similar parameter called STOP. In this case, the branching is not conducted when the button is pressed but it is conducted only when branching is permitted. In other words, it causes interrupt processing to be a temporarily stopped.

An interrupt processing routine can be set as follows:

```
ON MTRIG (1) GOSUB 100
```

This instruction means that the operation branches to the processing routine starting at Line No. 100 when Button A is pressed.

The following shows a program example.

(Example)

```
10 ON MTRIG (1)  GOSUB 100
20 ON MTRIG (2)  GOSUB 200
30 MTRIG (1)  ON
40 MTRIG (2)  ON
50 LOCATE 0, 0
60 PRINT USING "X=### Y=###" ;MREAD (0), MREAD (1)
70 GOTO 50
100 R=RND * 90+10
110 B=RND * 7
120 CIRCLE (MREAD (0), MREAD (1)),R,B
130 RETURN
200 C=RND * 7
210 PAINT (MREAD(0), MREAD(1)), C
220 RETURN
```

(4) **Graphic cursor**

The next explanation is on the GCURSOR statement which is closely related to control by the mouse.

In a GCURSOR statement, first, designate the first position to display the cursor and then describe two variables. An example is shown below:

```
GCURSOR (320, 100), (X, Y)
```

When the above is executed, a graphic cursor (a cross) appears at the center of the screen. Press the return key. The graphic cursor disappears and the computer returns to the command level. The coordinates at the time of pressing the return key can be obtained by the following instruction.

```
PRINT X ; Y
```

Execute the following program:

```
10  CLS
20  GCURSOR (320, 100), (X1,Y1), (X2, Y2)
30  LINE (X1, Y1)-(X2, Y2), PSET, 3
40  END
```

As shown by the above example, two or more variables to be read can be specified. The above example has only two of them, but up to 30 variables can be specified by delimiting each of them by a comma. Pressing the return key reads the cursor position at two positions, and a line connecting these two points is drawn.

The above explanation is on the basic use of the GCURSOR statement. Note that the mouse can also be used to control the graphic cursor.

This specification is added at the end of the GCURSOR statement as a parameter. For example, if Line No. 20 of the program shown above is changed as shown below, the graphic cursor can be controlled by the mouse:

```
20 GCURSOR (320, 100), (X1, Y1), (X2, Y2), "M"
```

In this case, trigger button A is used instead of the return key. Trigger button B sets the graphic cursor to the original position.

Kanji processing

As explained in the image generator section, the area where character patterns (alphabetic, Katakana and Hiragana characters) are memorized is called the character generator. These patterns are memorized in units of 8 x 8 dots. The problem we have with Kanji characters is that the majority of Kanji characters cannot be expressed by 8 x 8 dots. Therefore, another character generator having a greater number of dots is needed to express Kanji characters, and this is the Kanji ROM. The Kanji ROM is inserted into the S1 slot, like an interface card, and once it is inserted, the computer can use Kanji characters. S1 has statements with which Kanji characters can be expressed in BASIC when using the Kanji ROM.

The character pattern of Kanji characters use the size of 16 x 16 dots. Therefore, 20 x 12 characters can be displayed in the 40-character mode and 40 x 12 characters in the 80-character mode. The Kanji ROM contains 2,465 Kanji characters of JIS (Japanese Industrial Standards) and 453 non-Kanji character patterns.

(1) **KANJI statement**

The first method to display Kanji characters is the use of a KANJI statement. This instruction basically specifies the Kanji code. For example, when the following instruction is given:

```
KANJI &H467C
```

the character 「日」 is displayed on the upper left of the screen. As shown in the above example, all Kanji codes are specified in hexadecimal notation. For information on Kanji characters that can be displayed and their correspondence with Kanji codes, refer to the "Guide to BASIC".

Next, input the following:

```
KANJI &H4E29
```

The character 「立」 is displayed next to 「日」. Thus, a KANJI statement causes the display of Kanji characters in succession. Input the following to display the two characters at the same time:

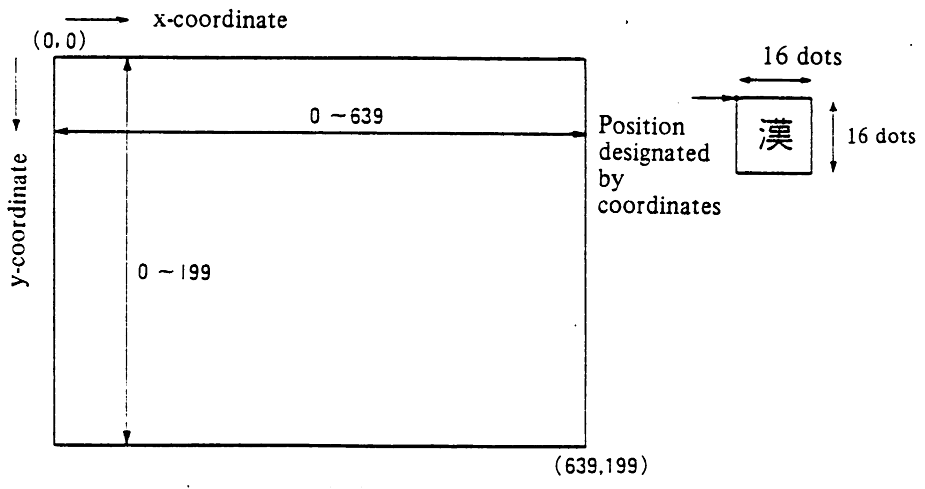
```
KANJI &H467C, &H4E29
```

As the example shows, to display two or more Kanji characters, describe the Kanji codes in succession but delimit each of them by a comma.

This shows how characters should be displayed starting at the home position, but what should be done to display characters in the lower part of the screen? In such a case, designate the coordinates of the position where they are to be displayed. Note that these coordinates are not character coordinates but are graphic coordinates. In other words, Kanji characters are displayed in a graphic pattern. Execute the following instruction:

```
KANJI (200, 100), &H467C
```

This time, the character 「日」 is displayed in the center of the screen. In this instruction, a comma is necessary between the ending parenthesis and Kanji code, and the values of these coordinates designate the position of the dot in the Kanji's upper left corner.



Coordinates on screen

(2) **KANJI@ statement**

Although we have stated that Kanji characters are displayed as graphic patterns, the KANJI statement cannot specify colors. The KANJI@ statement is provided to specify colors. The KANJI@ statement is basically the same as the KANJI statement, and the following instruction causes the same display as had before:

```
KANJI@ (200, 100), &H467C, &H4E29
```

Note that the delimiter between the parenthesis and Kanji code is a semi-colon instead of a comma. A color specification is given as shown below:

```
KANJI@ (200, 100), PSET, 4 ; &H467C, &H4E29
```

This instruction displays「日立」in green. Color specifications are made using palette numbers. Upon seeing this display, it might be thought that Kanji characters displayed colorfully are not easy to read, but this is up to the liking of individual users.

GUIDE FOR READING CHAPTERS 2 AND 3

Chapter 2 explains BASIC instructions and Chapter 3 BASIC functions. These explanations are given in the order TITLE, FUNCTION, FORMAT, STATEMENT EXAMPLE, EXPLANATION, NOTE, REFERENCE and SAMPLE PROGRAM. The following outlines precautions for reading these chapters.

TITLE

When a title is in an abbreviated form, the original full spelling is added in parentheses.

FUNCTION

This briefly explains the function of the particular instruction.

FORMAT

This shows the instruction description format. The following preconditions are set for the signs used here:

(1) Items written in capital alphabetic characters should be input as they are.

(2) Items written in < > must be specified by the user.
Example: NEW ON <expression>
Write a suitable constant or variable for the <expression>.

(3) Brackets [ ] mean that the description in the brackets may be omitted.
Example: RUN [<Line No.>]
The Line No. may be omitted. If a group of parameters are enclosed by brackets, description of all parameters may be omitted. However, if individual parameters are enclosed by brackets, at least one parameter must be specified.
Example: CLEAR [[<size of character area>] [,<head address of machine language area>]] .... In this case, description of all parameters may be omitted.
Example: COLOR [<character color>] [,[<back color>] [<IG switch>]] ...... In this case, specify at least one parameter.

(4) For items containing the abbreviation sign ⌜.....⌟, the item specification can be written as many times as the particular line can contain.
Example: DATA <constant> [, <constant>]...
In this case, the following description is permitted.
DATA <constant>
DATA <constant>, <constant>
DATA <constant>, <constant>, <constant>

(5) When written $\left|\begin{matrix};\\,\end{matrix}\right|$, it means that either the upper or lower sign can be used. This example shows that either a semi-colon ⌈;⌋ or comma ⌈,⌋ can be used. However, it must be noted that sometimes the operation may be different depending on the selection of the upper or lower sign.

(6) When a keyword has an abbreviated form, it is shown after the FORMAT and is enclosed in parentheses ⌈( )⌋. When parentheses are used in other cases, they are the necessary symbols for the format, and they must be written in a program.

Example: CONT (abbreviated form is C.)

In this case either C. or CONT may be used.

Also, when a statement is written in its abbreviated form in a program, all statements other than ⌈'⌋ (abbreviated form of REM) are changed to the full form and stored, which you can see by having the computer output a program listing.

(7) Specify coordinates as follows: (Wx, Wy) for world coordinates, (Sx, Sy) for screen coordinates, (Dx, Dy) for display coordinates, and (X, Y) for character coordinates.

(8) When a format is shown as ⌈$[\underset{①}{,}$ [<parameter 1>] [,<parameter 2>] ......⌋, it is not permitted to specify the comma ① alone and omit <parameter 1>, etc.

STATEMENT EXAMPLE

This shows an actual example of format description.

EXPLANATION

This explains the instruction contents in detail.

NOTE

Cautions to be paid are described here. If there is nothing, this column is not shown.

REFERENCE

Instructions that are related to the instruction are listed.

SAMPLE PROGRAM

An example of using the instruction is shown by an actual program.

$\boxed{\text{A}}$

AUTO command

FUNCTION	This command automatically generates line numbers.
FORMAT	AUTO [[< Line No.>] [, < increment>]]
STATEMENT EXAMPLE	AUTO 10,100

EXPLANATION

While this command is effective, each time the $\boxed{\lrcorner}$ key is pressed, line numbers are automatically generated, starting at the <Line No.> and increasing by the <increment>. When both <Line No.> and <increment> are omitted, "10" is used. If a comma 「,」 is specified and <increment> is omitted, the increment for the AUTO command that was executed immediately before is used.
A Line No. exceeding "63999" or a Line No. that is the same as a Line No. used in an existing program cannot be generated by this command.
To set the computer to the command level, input $\boxed{\text{BREAK}}$, $\boxed{\text{CTRL}}$ + $\boxed{\text{C}}$, or press the $\boxed{\lrcorner}$ key without inputting anything after the Line No. At this time, the line with the Line No. that was generated last is ignored.

NOTE

During execution of an AUTO command, if the $\boxed{\text{CLS}}$ key is pressed in the middle of inputting a program, the computer is set to the state in which it waits for a command.
During execution of an AUTO command, if the cursor is moved to other line and the $\boxed{\lrcorner}$ key is pressed, the other line can be corrected, but the computer is set to the state in which it waits for a command.

SAMPLE PROGRAM

```
Ready
AUTO ···························(1)
10 '
20 '
30

Ready
AUTO 100,50 ··········(2)
100 '
150 '
200

Ready
NEW

Ready
AUTO ,30 ···················(3)
10 '
40 '
70

Ready
```

(1) shows the state of line number generation when "AUTO" only is input. The <starting Line No.> and <increment> are set to 10 and 10.

(2) shows the state of line number generation when 100 is specified for the <starting Line No.> and 50 for the <increment>. The Line No. starts at 100 and increases in units of 50.

(3) shows the state of line number generation when 30 is specified for the <increment>. The <starting Line No.> is set to 10.

When executing (3) after executing (1), since (3) specifies 10 for the <Line No.>, execute a NEW command first before executing the AUTO command.

BEEP

FUNCTION	The built-in speaker emits a beep.
FORMAT	BEEP [<numeric value>]

EXPLANATION

When this command is executed, the speaker emits a beep. A numeric value may be specified after BEEP but it is ignored at the time of execution. The numeric value must be in the range from "0" to "255".
This is the same as 「PRINT CHR$ (7) ;」.

SAMPLE PROGRAM

```
Ready
LIST

100 ' BEEP command
110 FOR I=1 TO 5
120 BEEP
130   FOR J=1 TO 100
140   NEXT J
150 NEXT I
160 END

Ready
RUN

Ready
```

This is a program which causes the emission of interrupted sound. The FOR and NEXT statements on Line Nos. 130 and 140 are a time delay loop to wait for the beep generated by the BEEP command on Line No. 120 to die out. If this is repeated five times using the FOR ~ NEXT statements starting at Line No. 110, the beep sounds five times.

C

CIRCLE

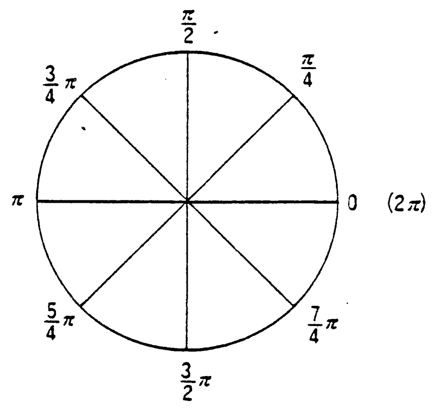
FUNCTION — Draws a circle or oval.

FORMAT — CIRCLE (<Wx, Wy>), <radius> [, [<color>]
[, [<starting angle>] [, [<ending angle>] [, [<ratio>]
[, [<plot option>]
[, <line style>]]]]]]

STATEMENT EXAMPLE — CIRCLE (100, 100), 10, 2

EXPLANATION

This command draws a circle with it center world coordinates <Wx, Wy> and with the specified <radius>. An oval is drawn at the specified <ratio>. Specify a Palette No. for the <color>. When it is omitted, the character color specified by the COLOR statement is used.
When the <starting angle> and <ending angle> are specified, an arc in the specified range is drawn. Use a radian value to specify the angle (in the range from "-2π" to "2π"; "-0" radian cannot be specified). When these specifications are omitted, "0" radian is set as the <starting angle> and "2π" radians to the <ending angle>.
When a negative value is specified as an angle, its absolute value is taken and the positive angle is used. However, the radius is drawn from the center in the direction of the negative value specification. A fan shape can be drawn by specifying negative values as the <starting angle> and <ending angle>.



Specify the <ratio> as the ratio of (radius in the vertical direction)/(radius in the horizontal direction).
When the <ratio> is smaller than 1, the <radius> means the radius in the horizontal direction. On the other hand, if it is larger than 1, it means the <radius> is in the vertical direction.

Specify the ratio as shown below to draw a circle or oval.

SCREEN statement / Shape	SCREEN 1 or omission	SCREEN 0
Oval longer in the horizontal direction	Ratio $< \frac{5}{12}$	Ratio $< \frac{5}{6}$
Perfect circle	Ratio $= \frac{5}{12}$	Ratio $= \frac{5}{6}$
Oval longer in the vertical direction	Ratio $> \frac{5}{12}$	Ratio $> \frac{5}{6}$

(When the ratio of vertical to horizontal directions is equal in the window and view port)

When the <ratio> specification is omitted, the <ratio> means the radius in the horizontal direction and a perfect circle is always drawn on the screen regardless of the view port specification.
Specify one of PSET, PRESET, AND, OR, XOR and NOT for the <plot option>. If omitted, PSET is taken.
When the <line style> is specified, the circle is drawn as a broken line or alternate long and short dashes depending on the specified value. If this specification is omitted, the circle is drawn with a continuous line.
For details of the <plot option> and <line style>, refer to the explanation of the LINE statement.
When a CIRCLE statement is executed, the LP (refer to 1.10 Coordinate systems) determines the central coordinates of the circle.

NOTE

When the <line style> is specified in the 640 x 200 dot mode, the line does not look clean. This is because, since the dot interval in the vertical direction is very different from the dot interval in the horizontal direction of the screen, the length of the line on the screen is changed by the tilt of each part of the circumference.

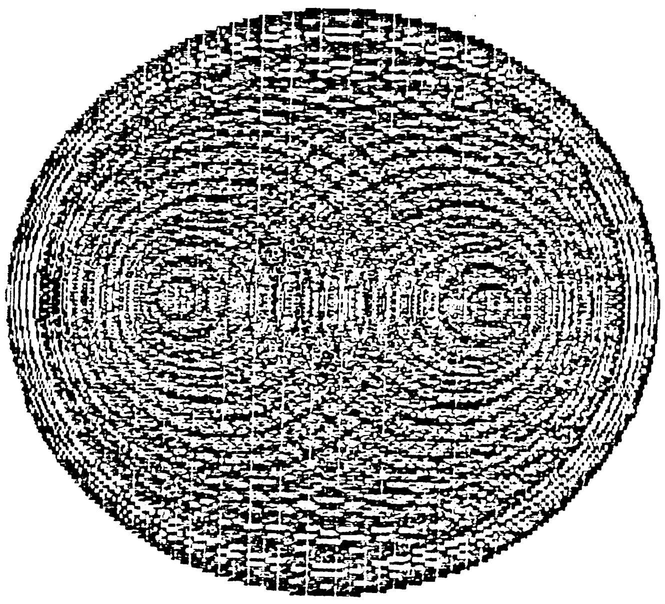
REFERENCE

COLOR

SAMPLE PROGRAM

```
Ready
LIST

100 'CIRCLE command
110 CLS:SCREEN 1
120 Y=100:N=50:I=1
130 FOR X=200 TO 380 STEP 4
140     R=N*SIN((X-200)*3.14159/180)
150     CIRCLE (X,Y),R,I
160 NEXT X
170 N=N+20:I=I+1:IF I=8 THEN 190
180 GOTO 130
190 END

Ready
```

```
Ready
LCOPY 4
```



This is an example of drawing graphics with a CIRCLE statement.
Line 110 erases the whole display on the screen and the SCREEN statement sets the graphics to the 640 x 200 dot mode. Line 120 assigns the initial value to the variables given to the parameters. Variable Y is the Y coordinate which is the center of circle drawn by the CIRCLE statement, and the specification is 100. Variable N is the maximum amplitude when radius R is changed by the sine curve, and variable I means the <attribute>. In this example program, 50 is specified to N and 1 is specified to I.
The FOR ~ NEXT statements on lines 130 to 160 draw a circle while increasing X from 200 to 380 in units of 4 and increasing or decreasing the circle radius as the sine curve. The action draws spherical graphics.
When one graphic is drawn, Line 170 changes each parameter. 20 is added to N and this enlarges the circle radius, resulting in a graphic that contains the graphic drawn earlier. Also, the I value increases by one and this changes the graphic color. When the I value becomes 8, the execution goes to Line 190 and the computer is set to the command level. Otherwise, the execution goes back to Line 130 and another graphic is drawn.

CLEAR

FUNCTION

Clears variables and assigns the memory to the character area and machine language area.

FORMAT

CLEAR [[<size of character area>] [,<machine language area top address>]]

STATEMENT EXAMPLE

```
CLEAR 500, 4000
```

EXPLANATION

A CLEAR statement releases all memories that are used as data while leaving the program that resides in the memory. The details are as follows.

(1) Changes all numeric variables to "0".

(2) Changes all character variables to " " (null string).

(3) Erases all arrays.

(4) All information defined by DEF statements (DEFFN, DEFINT, DEFDBL, DEFSNG, DEFSTR, DEFMAP) is made invalid.

(5) When the <size of character area> is specified, a character area of the specified size (in byte units) is secured. If this specification is omitted, the size remains unchanged. In the initial state (when BASIC is started or after executing a NEW ON statement), the size of the character area is set to 4094 bytes. Up to 45054 bytes can be specified.

(6) When the <machine language area top address> is specified, the area from the specified address to address &HCFFF is assigned to the memory as a machine language area. If this specification is omitted, the machine language area remains unchanged. In the initial state, no memory is assigned as the machine language area. Specify &HD000 to release all machine language areas that have been assigned.

SAMPLE PROGRAM

```
Ready
LIST

100 'CLEAR command
110 DIM M(3)
120 A= 6809:B$= "Personal Computer"
130 FOR I=0 TO 2
140     M(I) = I + 2
150     PRINT  "M(I) = ";M(I)
160 NEXT I
170 PRINT"A = ";A,"B$ = ";B$
180 PRINT
190 CLEAR
200 FOR I =0 TO 2
210     PRINT "M(I) = ";M(I)
220 NEXT I
230 PRINT"A =    ";A,"B$ = ";B$
250 END

Ready


RUN
M(I) =  2
M(I) =  3
M(I) =  4
A =  6809      B$ = Personal Computer

M(I) =  0
M(I) =  0
M(I) =  0
A =    0       B$ =

Ready
```

This is an example of initializing variables by a CLEAR statement.
Lines 110 to 170 assign data to each of A, B$ and Array M and they are displayed.
Line 190 executes the CLEAR statement.
Lines 200 to 230 display the contents of A, B$ and Array M once again.
The execution result indicates that all numeric variables have been changed to "0" and null strings have been assigned to character variables.

CLOSE

FUNCTION Closes input/output file(s).

FORMAT CLOSE [[#] <File No.> [, [#] <File No.>] ...]

STATEMENT EXAMPLE CLOSE #1

EXPLANATION

This command closes input/output files that use I/O buffers with the specified numbers and prepares the buffers with the specified numbers for use by other files. When a CLOSE statement is executed without specification of the <File No.>, all files that are open are closed. No error results even if a CLOSE statement is executed when all files are closed.
As well as this command, execution of an END statement or RUN command closes all files.

NOTE

When a file is in the output mode, if the file is kept in the open state when the file processing is completed, sometimes all data is not output to the output device.
Make sure to close the file with a CLOSE statement, etc., in order to output all data in the buffer from the buffer to the output device as a result of a program execution.
Also, in the case of an instruction in which a file is directly specified by a file descriptor, the file is kept in the open state during BASIC processing. When execution of the instruction is temporarily held by inputting $\boxed{\text{CTRL}}$ + $\boxed{\text{D}}$ keys, make sure to execute a CLOSE command.

REFERENCE

OPEN

SAMPLE PROGRAM

```
100 'CLOSE command
110 OPEN "O",#1,"TSTFILE1"
120 LINE INPUT"Enter string please.   :",A$
130 PRINT #1,A$
140 CLOSE #1
145 STOP
150 '
160 PRINT"FILE IS MADE"
170 OPEN "I",#1,"TSTFILE1"
180 LINE INPUT #1,A$
190 PRINT A$
200 CLOSE #1
210 END

Ready
RUN
Enter string please.   :"ABC"

Break In 145
Ready
CONT
FILE IS MADE
"ABC"

Ready
```

This program stores character strings input from the keyboard into a file on a cassette tape, reads and outputs them, and erases the file.
Line 110 opens "TSTFILE1" as a file for output, numbering it as File No. 1. The Line input statement on Line 120 reads the input from the keyboard and assigns the read result to A$. Line 130 makes a file of the contents of A$. The CLOSE statement on Line 140 closes "TSTFILE1". Since Line 145 sets the computer back to the command mode, the cassette tape is rewound. Line 160 shows that the file has been prepared. "TSTFILE1" is opened by Line 170 as an input file. At this time, since the OPEN statement on Line 110 is closed by the CLOSE statement on Line 140, the same File No. of "1" can be used. Line 180 reads the contents of the file, Line 190 displays the contents and Line 200 closes the file.

CLS (Clear screen)

FUNCTION	Clears the display on the screen and moves the cursor to the home position.
FORMAT	CLS [<Function No.>]
STATEMENT EXAMPLE	CLS 1 CLS

EXPLANATION

Erases the display's contents, specifying 0 ~ 3 as the <Function No.>.

<Function No.>	Contents
0	Erases the whole screen display (both graphic and text).
1	Erases only graphics from the display.
2	Erases only text from the display.
3	Erases only graphics in the current view port.

When the specification of the <Function No.> is omitted, the result is the same as specifying "0".
When "0", "1" or "3" is specified as the <Function No.>, LP (refer to 1.10 Coordinate systems) also returns to the home position (upper left corner of the screen). The display coordinates of Kanji also return to the reference point (0, 0).
When "0" or "2" is specified as the <Function No.>, the cursor returns to the home position (upper left corner of the screen).

REFERENCE

VIEW

COLOR

FUNCTION	(1) Specifies the character color and background color. (2) Switches the mode between the character code mode and IG mode.
FORMAT	COLOR [<character color>] [,[<background color>] [,<IG switch>]]
STATEMENT EXAMPLE	COLOR 4, 5

EXPLANATION

Specify a Palette No. for the <character color> as the color of characters to be displayed or for the color when the color specification is omitted in graphic instructions (LINE, CIRCLE, CONNECT, PAINT, KANJI@ and PUT@). When a number in a range of 8 ~ 15 is specified, the color is that of the Pallette No. with "8" subtracted from it, and the character is displayed in the reverse method.
Specify a Pallette No. as the <background color> as the background against which characters and graphics will be displayed.
Either a PALETTE or COLOR statement can be used to change the color. Note that a PALETTE statement is effective for all specifications of the <Palette No.> from the start of the program to the last PALETTE statement, but a COLOR statement is effective only for characters from the character immediately after executing the COLOR statement up to the character immediately before executing the next COLOR statement.
When a parameter is omitted in a COLOR statement, the value used before is applied.
Specifying "1" as the <IG switch> sets the IG mode. Omission of the <IG switch> or specifying "0" sets the character mode.
The character color displayed after executing a COLOR statement is changed immediately, but character colors and background colors that have already been displayed on the screen do not change. To make the color specified by a COLOR statement valid for the whole screen, erase the screen display by inputting "CLS 0" after executing the COLOR statement.

REFERENCE

PALETTE

NOTE

(1) Execute a POKE statement as shown below, to color the outer frame (outside of the display area). Note that if this is executed via RS-232C or a cassette tape, the color of outer frame becomes black.

```
POKE &HFFD0, <color code>
```

(2) Omission of all parameters of a COLOR statement is not permitted.

SAMPLE PROGRAM

```
LIST

100 ' COLOR statement
110 FOR I=1 TO 7
120   COLOR I-1, I
130   CLS
140   FOR K=1 TO 24: PRINT TAB(K);"HITACHI": NEXT K
150   FOR J=1 TO 1500: NEXT J
160 NEXT I
170 COLOR 7, 0
180 CLS
190 END

Ready
```

```
      HITACHI
       HITACHI
        HITACHI
         HITACHI
          HITACHI
           HITACHI
            HITACHI
             HITACHI
              HITACHI
               HITACHI
                HITACHI
                 HITACHI
                  HITACHI
                   HITACHI
                    HITACHI
                     HITACHI
                      HITACHI
                       HITACHI
                        HITACHI
                         HITACHI
                          HITACHI
                           HITACHI
                            HITACHI
                             HITACHI
```

This is an example of a program that changes the character and background colors using a COLOR statement.

The specifications of FOR ~ NEXT on lines 110 through 160 change "I" from "1" to "7" and the <character color> of the COLOR statement specifies "I-1" and the <background color> specifies "I". Line 140 displays a character and Line 150 causes the display to be delayed.

These are repeated seven times, and the color and background colors are displayed as changing colors.

COM(n) ON/OFF/STOP

FUNCTION	This permits, prohibits or stops interrupts from a communication port.
FORMAT	COM (<Port No.>) ON / OFF / STOP
STATEMENT EXAMPLE	COM (1) ON

EXPLANATION

This controls the input of interrupts of communication ports. Specify a value in a range from "0" to "4" as the <Port No.>.
This instruction is concurrently effective for the five communication ports (COM 0: ~ COM 4:).
COM(n) ON permits interrupts because of input made via the communication port specified by the <Port No.>. Once this instruction is executed, control is transferred to the line specified by a ON COM(n) GOSUB statement.
COM(n) OFF prohibits interrupts caused by inputs from the communication port. Once this instruction is given, interrupts caused by input are ignored.
COM(n) STOP causes a temporary stop of the acceptance of interrupts caused by inputs. Once this instruction is given, control is not transferred to the processing program, but the occurrence of the input is kept in memory. If COM(n) ON is executed after this, control is transferred to the specified processing program. A COM(n) STOP statement is effective in the COM(n) ON state.
Any data received in the COM(n) STOP state is stored in the buffer, as in the COM(n) ON state.

NOTE

Execution of a CLEAR, NEW or RUN statement sets the computer to the COM(n) OFF state.

REFERENCE

ON COM(n) GOSUB

SAMPLE PROGRAM

```
LIST

100 'COM(n) ON/OFF/STOP sratement
110 ON COM(0) GOSUB 160
120 COM(0) ON:I=0
130 OPEN "I",#1,"COM0:(F8N2)"
140 PRINT"WATING FOR COM INPUT"
150 GOTO 150
160 'INTERRUPT ROUTINE
170 I=I+1
180 INPUT #1,A$
182 PRINT A$;
190 IF I=100 THEN PRINT"END":CLOSE #1:END
210 RETURN

Ready
```

This shows an example of reading an input from a communication port. Line 110 defines the starting Line No. of the interrupt processing routine, and execution of Line 120 permits interrupts from Communication Port 0. Because of this statement, an interrupt is output each time after OPEN, and execution is transferred to Line 160.
Line 130 opens Communication Port 0.
An input is waited for in the loop of Line 150.
As an input occurs, Line 170 checks the end count, and if it is the end, the computer returns to the command level. If it is not the end, the input of the communication port is assigned to AS and the input is displayed on the screen.

CONNECT

FUNCTION — Connects the specified points by a straight line.

FORMAT

```
CONNECT [(<Wx1, Wy1>)] – (<Wx2, Wy2>)
   [–(<Wx3, Wy3>)] ...
[,[<color>] [,[<plot option>] [,<line style>]]]
```

STATEMENT EXAMPLE

```
CONNECT (100, 100) – (120, 110), 5, PSET
```

EXPLANATION

Increments "n" positions are connected by straight lines.
The <color> shows the color of the line. If this is not specified, the character color specified in the COLOR statement executed immediately before is used.
Specify any one of PSET, PRESET, AND, OR, XOR, and NOT for the <plot option>. If it is omitted, PSET is assumed. For details, refer to the explanation of the LINE statement.
When the specification of <Wx_1, Wy_1> is omitted, the line is drawn from LP (refer to 1.10 Coordinate system).
The number of positions that can be specified is up to 30 including LP that is assumed when the specification of <Wx_1, Wy_1> is omitted.
The specification of the last <Wx_n, Wy_n> becomes a new LP.
A dotted line or a line of alternate long and short dashes is drawn depending on the specified value of <line style>. When the specification of <line style> is omitted, a continuous line is drawn. For details of the specification method, refer to the explanation of the LINE statement.
The major difference between the case of drawing two or more continuous lines by a CONNECT statement and the case of doing it by combining LINE statements is than in the former case, the <line style> becomes continuous. In other words, in the case of a LINE statement, even if the <line style> is specified, the line is not necessarily continuous since the <line style> is applied from the beginning each time. In the case of a CONNECT statement, the straight line is drawn as continuous after adjusting the <line style>. This is a very convenient function when drawing a curved line graph.
However, in the 640 x 200 dot mode, since the intervals both in the vertical and horizontal directions become extremely different, if the tilt changes too much, the line looks as if it is not continuous.

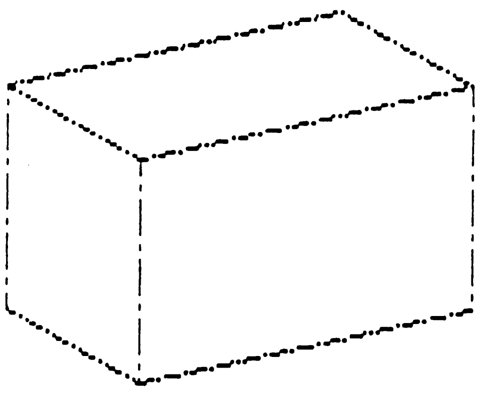
REFERENCE

LINE

SAMPLE PROGRAM

```
LIST

10 ' CONNECT statement
20 CLS
30 CONNECT (250,70)-(450,50)-(370,30)-(170,50)-(250,70),4,,&HF99F
50 CONNECT -(250,130)-(250,130)-(450,110)-(450,50),4,,&HF99F
60 CONNECT (250,130)-(170,110)-(170,50),4,,&HF99F
100 END

Ready
```

```
Ready
LCOPY
```



This draws graphics in green using line of alternate long and short dashes as specified by Lines 120 through 170.

CONSOLE

FUNCTION Specifies a console window.

FORMAT CONSOLE <scroll window starting line>, <number of lines to scroll> [,<function key display switch>] (The abbreviated form is CONS.)

STATEMENT EXAMPLE CONSOLE 10, 5

EXPLANATION

This specifies a range by which the screen should be scrolled.
The term scroll means that, when a display fills the screen, the display contents are moved up by one line in order to allow the display of an additional line.
Any number in the range from "0" for the top line to "24" for the bottom line can be specified for the <scroll window starting line>.
When the <function key display switch> is specified as "1", the character string registered for the programmable function keys is displayed on the bottom line of screen This function can be released by specifying "0". When this specification is omitted. the preceding state continues.

NOTE

When this instruction is executed, the cursor is set to the upper left of the scroll window.
The total of the three parameters must not exceed "25".

REFERENCE

WIDTH

SAMPLE PROGRAM

```
Ready
LIST

100 'CONSOLE statement
110 CLS
120 CONSOLE 10,3,0
130 FOR I=1 TO 1000
140 PRINT I;
150 NEXT I
160 END

Ready

 998  999  1000
Ready
```

Lines 120 through 140 display the values from "1" to "1000" starting at the 10th line. of the screen and scroll at intervals of three lines.
The execution result shows that is has been completed.

CONT (Continue command)

FUNCTION — Restarts the execution of a program that is temporarily stopped.

FORMAT — CONT (The abbreviated form is C.)

EXPLANATION

After input of $\boxed{\text{BREAK}}$ or $\boxed{\text{CTRL}}$ - $\boxed{\text{C}}$ keys or executing STOP and END statements, if CONT is input when the computer is waiting for a command, the execution restarts from the statement next to the statement where execution stopped. If the stop was caused by an INPUT statement, execution starts from the INPUT statement.
However, if the program is edited (added, corrected or deleted) after stopping, execution cannot be continued by a CONT statement.

NOTE

When the $\boxed{\text{BREAK}}$ or $\boxed{\text{CTRL}}$ + $\boxed{\text{C}}$ keys are pressed to temporarily stop during execution of FOR ~ NEXT statements, the execution cannot be restarted by a CONT statement in some cases.
When an END statement is executed, files are closed.

REFERENCE

STOP, END

SAMPLE PROGRAM

```
Ready
LIST

100 'CONT command
110  PRINT "MB-S1/10 !!!!!"
120  '
130 STOP
140 '
150 FOR I=1 TO 2
160 PRINT"Personal Computer":FOR J=1 TO 1000:NEXT J
170 NEXT I
180 END

Ready
RUN
MB-S1/10 !!!!!

Break In 130
Ready
CONT
Personal Computer
Personal Comouter

Ready
```

This is an example of a program that shows how to use the CONT statement.
When this program is executed, the program's execution is temporarily stopped by the STOP statement on Line 130. Execution can be restarted from the program line next to the program line where it stopped by inputting a CONT statement.

DATA

FUNCTION Defines data used by READ statements.

FORMAT DATA <constant> [,<constant>] ...

STATEMENT EXAMPLE DATA A, 213, "X, Y", &H100

EXPLANATION

This statement stores <constants> ready for READ statements.
The DATA statement is a non-executed statement, and as shown in the example of the READ statement, it can be written freely in a program, as many times as is needed. One DATA statement can have a number of the <constants> as long as they can be contained on one line and each of them is delimited by a comma. READ statements sequentially read the constants (within 255 characters) starting at the DATA statement with the highest Line No. A DATA statement on any line can be specified by a RESTORE statement.
For the <constant> in a DATA statement, a numeric constant or Hollerith constant of any type can be used, provided that a Hollerith constant including a comma or colon or having a blank space before and after it is identified by double quotation marks「"」. Use of double quotation marks is unnecessary in other cases, and even if it is used, it is ignored.

NOTE

No error results if a DATA statement is input as a command, but it cannot be read by a READ statement.

REFERENCE

READ, RESTORE

SAMPLE PROGRAM

```
Ready
LIST

100 'DATA statement
110 S=0
120 FOR I=1 TO 4
130       READ SAM
140          S = S + SAM
150 NEXT
160 PRINT S
170 '
180 READ A$,B$
190 PRINT A$,B$
200 '
210 DATA 580,1350,1350,1030
220 DATA Personal Computer ,6809
230 END

Ready
RUN
 4310
Personal Computer             6809

Ready
```

This is an example of reading numeric data and character strings using DATA statements. The FOR ~ NEXT statements starting at Line 120 reads and adds the numeric values on Line 120, and the result remains in Variable S. This arrangement of writing calculation data in a DATA statement and reading them by a READ statement at the point of calculation is very convenient since a change in the data requires a change of only the DATA statement.

DEF FN (Define function)

FUNCTION — Defines a user function.

FORMAT — DEF FN <name> [(<dummy argument> [,<dummy argument>]...)] = <definition expression of function>

STATEMENT EXAMPLE — DEF FNRMS (X, Y) = SQR (X∧2 + Y∧2)

EXPLANATION

This statement defines a function.
「FN <name>」 is used as the function name and the <name> must satisfy the conditions required of a variable name (refer to 1.7 Variables).
The description for the <dummyargument> must also satisfy the conditions required of a variable name, but since this specification is used to show the number of arguments and their type (whether they are numeric or character) and the type of the <definition expression of function>, using the same variable name at two or more places in one program creates no problem in operation.
The <definition expression of function> shows the operation contents of the function, and it must be written within one line (up to 255 characters). The dummy argument used in this expression is used after being replaced by a real argument at the time the function is quoted. For any variable which does not concur with any <dummy argument> used on the statement, the value of the variable when it is quoted is assigned as is for the operation.
The type of function is determined by the attribute character of the <name>, and if there is no attribute character, it forms a single-precision type. However, if a definition has been given to the starting character of the <name> by a DEFINT/SNG/DBL/STR statement, the function name without an attribute character at its start takes the type as defined.
Definition of a function must be given before it is quoted. A definition statement can contain any other user function(s).
The operation precision of a function is determined by the type of function and its definition expression, and it is not related to the type of real argument.
(Example)

```
10 DEF FNA# (X#, Y#) =X#/Y# : P!=1 : Q!=3
20 DEF FNB# (X ,Y) =X/Y : P#=1 : Q#=3
30 PRINT FNA# (P! , Q!)
40 PRINT FNB# (P# , Q#)
RUN
 .3333333333333333
 .3333333432674408
Ready
```

NOTE

When there is an error in a DEF FN statement, the error message points out the line where the function is quoted.

REFERENCE

FN function

SAMPLE PROGRAM

```
Ready
LIST

100 'DEF FN statement
110 DEF FNAREA(R)=3.14159*R^2
120 '
130 FOR I=1 TO 4
140        READ R
150        PRINT FNAREA(R);
160 NEXT I
170 DATA 2,6,10,1
180 END

Ready
RUN
 12.5664   113.097   314.159   3.14159
Ready
```

This shows an example of defining a function by a DEF FN statement. The DEF FN statement on Line 110 defines a function called FNAREA having a dummy argument of R in order to calculate the area of a circle. R is the radius of the circle. The function defined here is quoted on Line 150. For the dummy argument, R is used as used in the DEF FN statement, but a different variable may be used.
By defining a function using a DEF FN statement, a non-intrinsic function or very complicated function can be created and used as many times as needed.

DEFINT/SNG/DBL/STR (Define integer/single/double/string)

FUNCTION	Defines the type of a variable with a given name.
FORMAT	DEF <type> <range of variable name starting characters> [,<range of variable starting character>] ...
STATEMENT EXAMPLE	DEFINT X, Y, A–D

EXPLANATION

This statement defines the type of variable name and array name that starts with the specified character. However, if an attribute character like「%」,「!」,「#」or「$」is given a variable name, this type has priority over the type defined by this statement.
Specify one of the following for the <type>.

INT	Integer type
SNG	Single-precision type
DBL	Double-precision type
STR	Character type

Placing a space between DEF and the <type> generates a syntax error.
The specification of the <range of variable name starting characters> should be made by delimiting one character or two characters that are connected by「–」(including the characters between the two in alphabetic order).
(Example)

A, C	This specifies A and C
A–C	This specifies A, B and C
A, C–F	This specifies A, C, D, E and F

If the two characters connected are not in alphabetic order, it generates a syntax error.
The variable of any variable name which starts with a character the type of which has not been defined and which has no attribute character is regarded as the single-precision type in all cases.
The type declaration statement becomes invalid when the contents of variables are all erased.

SAMPLE PROGRAM

```
LIST

100 'DEFINT/SNG/DBL/STR statement
110 '
120 DEFDBL X,Y
130 DEFSTR A-C
140 '
150 X=1#/3:Y=2#/3
160 A=" How ":B="are ":C="you."
170 PRINT"X=";X,"Y=";Y
180 PRINT A+B+C
190 END

Ready
RUN
X= .3333333333333333          Y= .6666666666666667
 How are you.

Ready
```

Line 120 declares double precision for variables X and Y and Line 130 declares character type for variables A, B and C. By these statements, the values of X and Y become double precision even though they are not written as X# and Y#, and A, B and C are handled as character variables even though they are not written as A$, B$, and C$.
As the execution result shows, Line 150 is calculated in double precision and variables A, B and C on Line 160 are assigned to character strings.

DEF MAP (Define map)

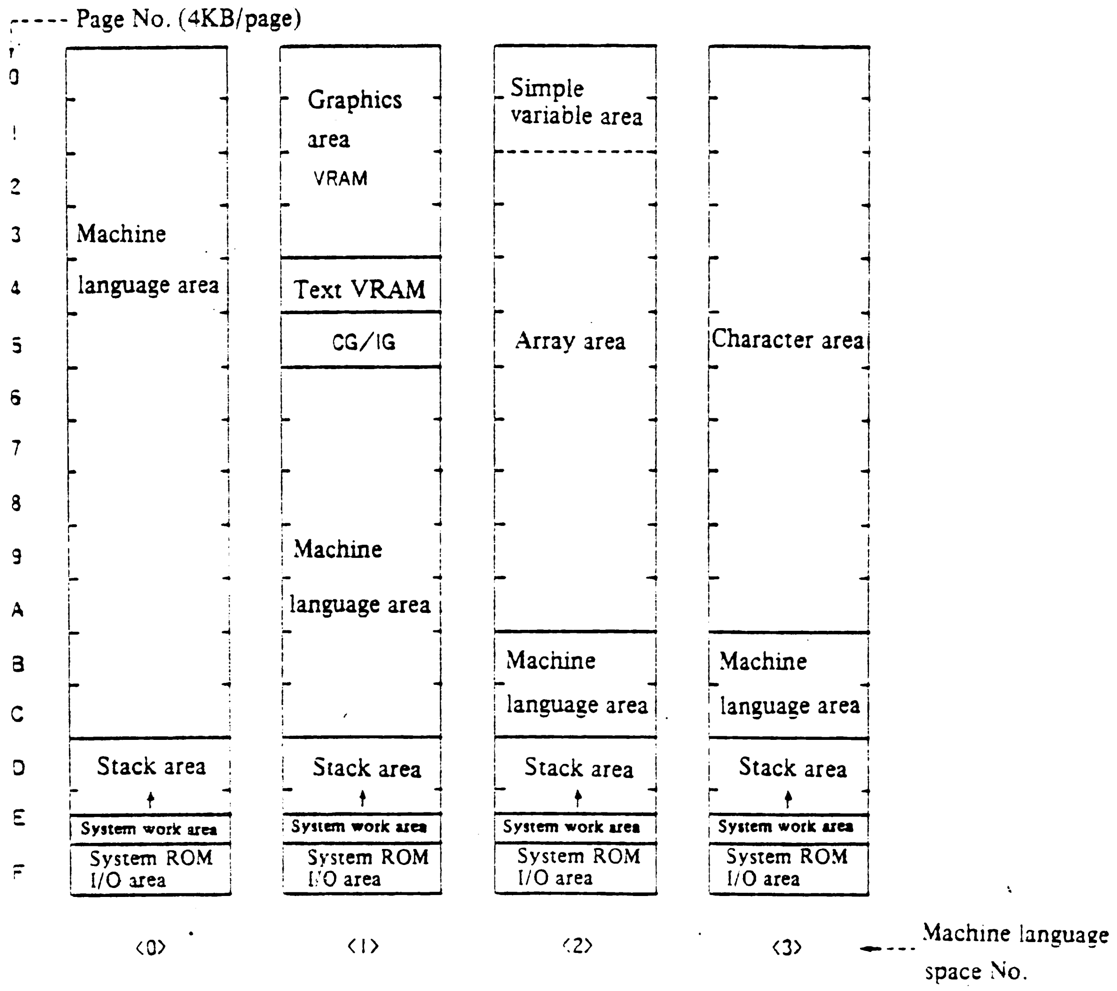
FUNCTION — Specifies a machine language space.

FORMAT — DEF MAP [= <machine language space No.>]

STATEMENT EXAMPLE — DEF MAP = 2

EXPLANATION

This statement specifies a machine language space used by machine language functions (MON, LOADM, SAVEM, EXEC, POKE statements and PEEK and USR functions). A DEF MAP statement selects one of the four languages spaces ranging from "0" to "3" and stores a machine language program in the selected machine language space of the user area. The switching to the selected space is conducted at the time of machine language function execution, and when the processing terminates, it is returned to the initial space. Execute RTS (machine language return instruction) to return from a machine language program to BASIC. When the specification of <machine language space No.> is omitted or when no DEF MAP statement is executed, the machine language space is set to "0".



NOTE

Specify the capacities of the character area and machine language area with a CLEAR statement.

REFERENCE

CLEAR, POKE and PEEK function

SAMPLE PROGRAM

```
Ready
LIST

100 'DEF MAP statement
105 CLEAR ,&HC000
110 DEF MAP=0
130 LOADM "TEST"
140 EXEC &HC000
150 END

Ready
```

This is an example of specifying machine language space "0" on Line 110, to expand the machine language area, and the machine language program is executed by Line 140.

DEF USR (Define user)

FUNCTION	Defines a starting address of a machine language subroutine.
FORMAT	DEF USR [<No.>] = <starting address>
STATEMENT EXAMPLE	DEF USR1 = &H400

EXPLANATION

This statement sets the execution start address of a machine language subroutine that is called by a machine language user function in the range from USR0 to USR9.
Specify "0" as the <No.>. When the specification is omitted, "0" is set.

REFERENCE

USR function

SAMPLE PROGRAM

```
LIST

100 'DEF USER statement
110 CLEAR 300,&HC000
120 DEF USR0=&HC000
130 DEF USR1=&HC9FF
140 POKE &HC000,&H39
150 POKE &HC9FF,&H39
160 C%=50:D$="ABC"
170 G%=USR0(C%):H$=USR1(D$)
180 PRINT G%,H$
190 END

Ready
RUN
 50             ABC

Ready
```

Since machine language subroutine written in the memory by Lines 140 and 150 does not do anything, the USR function on Line 170 returns the argument as the function value as it is.

DELETE command

FUNCTION — Deletes specified lines.

FORMAT

(1) DELETE <starting line No.> [$\left|\begin{matrix}-\\,\end{matrix}\right|$ [<ending line No.>]]

(2) DELETE $\left|\begin{matrix}-\\,\end{matrix}\right|$ <ending line No.>

STATEMENT EXAMPLE — DELETE 100–200

EXPLANATION

Format (1) deletes the program lines from the <starting line No.> to the <ending line No.>. If the <starting line No.> only is specified, that line only is deleted. If, however, the No. is accompanied by 「.」 or 「–」, all lines of the program from the specified <starting line No.> are deleted. In the case of Format (2), all lines from the program start to the specified line are deleted.
A period 「.」 may be used instead of each line No. If this is done, point 「.」 means the current line in BASIC memory resulting from the occurrence of an error or editing. When a DELETE command appears in a program, the computer is set to the state of waiting for a command after executing the DELETE command. All line numbers may be ones that actually do not exist, except if only the <starting line No.> is specified, it must be an existing No.

SAMPLE PROGRAM

```
Ready
LIST

100 'LINE-1
110 'LINE-2
120 'LINE-3
130 'LINE-4  ..................(1)
140 'LINE-5
150 'LINE-6
160 'LINE-7
170 'LINE-8

Ready


Ready
DELETE 110

Ready
LIST

100 'LINE-1
120 'LINE-3
130 'LINE-4
140 'LINE-5  ..................(2)
150 'LINE-6
160 'LINE-7
170 'LINE-8

Ready


Ready
DELETE 120-150

Ready
LIST

100 'LINE-1
160 'LINE-7  ..................(3)
170 'LINE-8

Ready


Ready
DELETE -160

Ready
LIST

170 'LINE-8 ......................(4)

Ready
```

This is an example of a DELETE statement. Suppose that the contents as shown by (1) are instructed first on Lines 100 to 170, in steps of 10 lines.
When DELETE 110 is executed, Line 110 is deleted as shown in (2).
The result of executing DELETE 120-150 in the (1) state is shown in (3). Four lines from Line 120 to Line 150 are deleted.
(4) shows the result of executing DELETE−160 in the (1) state. Lines from the start to Line 160 are deleted.

DIM (Dimension)

FUNCTION	Specifies the maximum number of elements in arrays and allocates memory areas.
FORMAT	DIM <array name> (<max. value of subscript> [,<max. value of subscript>] ...) [,<array name> (<max. value of subscript> [,<max. value of subscript>] ...)] ...
STATEMENT EXAMPLE	DIM A(10, 10), B$(100)

EXPLANATION

This statement sets the dimensions of arrays and the maximum values of subscripts, and at the same time, it allocates the memory area to these arrays. Since the minimum value of a subscript is always "0", the number of elements in one array is the product of the <max. value of subscript> plus one for each dimension.
For example, in the case of A (2, 3, 1), the number of elements is (2 + 1) x (3 + 1) x (1 + 1), which comes to 24.
When an array is used without declaring it in a DIM statement, the result is the same as declaring a DIM statement with the dimension number used for the first time and "10" as the maximum value of the subscript.
The dimension number and the maximum value of the subscript can be specified as the maximum values that the memory can allow.

SAMPLE PROGRAM

```
Ready
LIST

100 'DIM statement
110 DIM A(2,2)
120 FOR I=0 TO 2
130       FOR J=0 TO 2
140          READ A(I,J)
150       NEXT J
160 NEXT I
170 PRINT "READ  END"
180 FOR I=0 TO 2
190       FOR J=0 TO 2
200          PRINT "DATA";"(";I;",";J")";A(I,J)
210       NEXT J
220 NEXT I
230 DATA 1,2,3,4,5,6,7,8,9
240 END

Ready


RUN
READ  END
DATA( 0 , 0 ) 1
DATA( 0 , 1 ) 2
DATA( 0 , 2 ) 3
DATA( 1 , 0 ) 4
DATA( 1 , 1 ) 5
DATA( 1 , 2 ) 6
DATA( 2 , 0 ) 7
DATA( 2 , 1 ) 8
DATA( 2 , 2 ) 9

Ready
```

This program declares a 3 x 3 two dimensional array, and writes and displays the data. The size of array is declared by a DIM statement before the array is used. The FOR ~ NEXT statements, which are the nest on Lines 120 to 160, set the subscripts of the array to "I" and "J", and they are changed from "0" to "2" by each of them, and the data is input to the array by a READ statement. When everything has been input. Line 160 signals the end of input and displays the contents of the array.

EDIT command

FUNCTION — Displays specified line and moves the cursor to the head of the statement.

FORMAT — EDIT <Line No.>

STATEMENT EXAMPLE — EDIT 50

EXPLANATION

This statement displays the line with the specified <Line No.> and moves the cursor to the head of the first statement on the line. The line with the specified <Line No.> must actually exist.
When an EDIT command is executed after temporarily stopping the execution of a program because of an error. execution is for the line where the program stopped.

REFERENCE

LIST, DELETE

SAMPLE PROGRAM

```
Ready
LIST

100 'EDIT command
110 '    sample program
120 FOR I=1 TO 5
130      PRINT "BASIC"
140 NEXT J
150 END

Ready
RUN
Next Without For In 140
Ready
EDIT 140
140 NEXT J
```

This is an example of correcting a program using an EDIT command.
When this program is executed, since the <variable> of the NEXT statement on Line 140 does not concur the <variable> of the FOR statement on Line 140, an error of NEXT without FOR occurs on Line 140, and the computer returns to the waiting for a command state. Line 140 can be corrected by inputting "EDIT 140".

END

FUNCTION Terminates execution of a program.

FORMAT END

EXPLANATION

This statement terminates execution of a program and closes all files. The computer returns to the waiting for a command state.
The END statement can be written in the place where the program execution should be terminated and two or more END statements can be input. The last END statement at the program end may be omitted, but if it is omitted, the files are not closed.

REFERENCE

STOP

SAMPLE PROGRAM

```
Ready
LIST

100 'END statement
110 I=1
120 IF I=10 THEN END
130 PRINT TAB(I);"HITACHI"
140 I=I + 1
150 GOTO 120

Ready
RUN
 HITACHI
  HITACHI
   HITACHI
    HITACHI
     HITACHI
      HITACHI
       HITACHI
        HITACHI
         HITACHI

Ready
```

An END statement can be given in any place. In this program, the END statement is executed and the computer returns to the waiting for a command state only when the result of the IF statement on Line 120 is true.

ERROR

FUNCTION Generaties an error.

FORMAT ERROR <error code>

STATEMENT EXAMPLE ERROR 100

EXPLANATION

This statement generates an error. In other words, when an ERROR statement is executed, the processing is conducted in the same way as if an error that corresponds to the <error code> (1 ~ 255) had occurred. Refer to the Error Messages for the types and meanings of error codes.
Each user can define his own error code by using an undefined error code in an ERROR statement and use it in an error processing program.

SAMPLE PROGRAM

```
Ready
LIST

100 'ERROR statement
110 ON ERROR GOTO 500
120 FOR I=-1 TO 1
130 FOR J=5 TO -5 STEP -5
140 IF I=0 AND J=0 THEN ERROR 200
150 K=I/J
160 PRINT I;J;K
170 NEXT J
175 NEXT I
180 END
500 REM --- ｴﾗｰ ｼｮﾘ ﾙｰﾁﾝ ---
510 IF ERR=200 THEN K=1:RESUME 160
520 IF ERR=11 THEN PRINT I;J;" NG":RESUME 170
530 ON ERROR GOTO 0

Ready


RUN
-1  5 -.2
-1  0  NG
-1 -5  .2
 0  5  0
 0  0  1
 0 -5  0
 1  5  .2
 1  0  NG
 1 -5 -.2

Ready
```

Line 140 defines an Error Code 200 error by combining it with an IF statement. Line 510 in the error processing routine checks whether or not the error is that of the Error Code 200 type.

EXEC (Execute)

FUNCTION	Starts executing a machine language program.
FORMAT	EXEC [<starting address of a machine language program>]
STATEMENT EXAMPLE	EXEC &H7800

EXPLANATION

This statement transfers the control to a machine language program residing in the memory. If the starting address is specified, the machine language program is executed from this address. If the starting address specification is omitted, execution of the machine language program starts at the entry pointer (starting address specified when SAVEM was executed) of a program that was loaded immediately before from an external file or at the starting address specified by an EXEC command that was executed immediately before. The <starting address of a machine language program> can be specified in an expression.

REFERENCE

CLEAR, POKE, PEEK function, DEFMAP

SAMPLE PROGRAM

```
LIST

100 'EXEC statement
110 CLEAR 300, &HC000
120 FOR I=&HC000 TO &HC007
130   READ A
140   POKE I,A
150 NEXT
160 EXEC &HC000
170 END
180 DATA &H86, &H03, &HC6, &H06, &H3D, &H44, &H56, &H39

Ready
RUN

Ready
```

Line 130 reads the data specified by Line 180 and Line 140 writes the data to memory. When Line 160 executes it, the machine language stored in &HC000 is executed, and operation is transferred to Line 170.

FILES command

FUNCTION	Displays the catalog of files recorded on a cassette tape on the screen.
FORMAT	FILES ["CAS0:"]
STATEMENT EXAMPLE	FILES "CAS0:"

EXPLANATION

This statement displays the catalog of files recorded on a cassette tape on the screen. The catalog is displayed in the following format:

<File name> <Type> <Form>

The type is expressed as a numeric "0", "1" or "2", and they mean a BASIC program, data and a machine language program, respectively. For the form, "A" is used to show character form and "B" for internal expression form.

NOTE

The end of a cassette tape cannot be detected by a FILES command. Input CTRL + D to stop the operation.

REFERENCE

LFILES

SAMPLE PROGRAM

```
Ready
LIST

100 ' FILES statemnt
110 FILES
130 END

Ready


RUN

GET-PUT    0 A
INPUT-W    0 A
IG$        0 A
IMAGE      0 A
INTERVAL   0 A
INT-ON     0 A
KANJI      0 A
GCURSOR    0 A
Abort In 110
Ready
```

This is an example of executing a FILES command as a statement. When Line 110 is executed, the catalog names recorded on the cassette tape are displayed.

FOR ~ NEXT

FUNCTION Repeatedly executes a program.

FORMAT

```
FOR <Variable name> = <initial value> TO <end
  value> [STEP <increment>]
        .
        .
        .
NEXT [<variable name> [,<variable name>] ...]
```

STATEMENT EXAMPLE

```
FOR I= 0 TO 100 STEP 2
  ⁝
NEXT I
```

EXPLANATION

The contents of lines from the FOR statement to the NEXT statement are executed for the specified number of times. The variable in <variable name> is used as the counter.

(1) The <initial value> is assigned to the variable.

(2) The variable value and the <end value> are compared.

a) If the variable value is greater than the <end value> (smaller when the <increment> is a negative value), the execution moves to the statement after the NEXT statement.

b) If the variable value is smaller than the <end value> (greater when the <increment> is a negative value), the execution moves to the statement after the FOR statement and statements up to the NEXT statements are executed.

(3) The <increment> is added to the variable value by the NEXT statement, execution moves to the FOR statement and the comparison of Step (2) is conducted. This is called a FOR ~ NEXT loop.

An expression (other than a character expression) can be used as the <initial value>, <end value> and <increment>. When the specification of 「STEP <increment>」 is omitted, "1" is assumed as the increment.

A double-precision variable cannot be used as the <variable name>.

FOR ~ NEXT loops can be nested. In other words, another FOR ~ NEXT loop can be formed in one FOR ~ NEXT loop and this can be further multiplexed. One FOR ~ NEXT loop must be completely inside another FOR ~ NEXT loop. The FOR and NEXT statements must have a 1:1 correspondence with each other.

(Example)

```
Ready
LIST

10 FOR I=1 TO 2
20 FOR J=1 TO 4 STEP 2
30 FOR K=5 TO 3 STEP -2
40 A=A+1
50 PRINT "I=";I, "J=";J, "K=";K, "A=";A
60 NEXT K
70 NEXT J
75 PRINT
80 NEXT I
90 PRINT "I=";I, "J=";J, "K=";K
100 END

Ready
RUN
I= 1           J= 1           K= 5           A= 1
I= 1           J= 1           K= 3           A= 2
I= 1           J= 3           K= 5           A= 3
I= 1           J= 3           K= 3           A= 4

I= 2           J= 1           K= 5           A= 5
I= 2           J= 1           K= 3           A= 6
I= 2           J= 3           K= 5           A= 7
I= 2           J= 3           K= 3           A= 8

I= 3           J= 5           K= 1

Ready
```

This is an example of nested FOR ~ NEXT loops.
Each variable in the FOR ~ NEXT loops is displayed. "A" represents the number of loops. Line 90 shows the value of each variable after coming out of the FOR ~ NEXT loop. Note that they are greater than the end values.
When two or more FOR ~ NEXT loops end at the same place as shown by Lines 60 and 70 of the (Example), the whole thing can be represented by one NEXT statement by delimiting the specification of <variable name> using a comma 「,」. The contents of Line 60 in the example can be rewritten as follows:

60 NEXT K, J ⏎ (「NEXT J, K」 is not permitted.)
DELETE 70 ⏎

If there is only one <variable name> for one NEXT statement, the specification of <variable name> can be omitted. 「I」 of the <variable name> on Line 80 of the example can be omitted.

NOTE

A NEXT statement must be written after a FOR statement not only in the order of execution but in the order of the program.
Also, one FOR statement cannot have two or more NEXT statements.

REFERENCE

WHITE ~ WEND

$\boxed{\mathbf{G}}$

GCURSOR (Graphic cursor)

FUNCTION	Reads the dot coordinates of the graphic cursor on the screen.
FORMAT	GCURSOR (<Wx, Wy>), (<variable 1>, <variable 2>) [, (<variable 3>, <variable 4>)]... [, <device name>]
STATEMENT EXAMPLE	GCURSOR (50, 25), (X1, Y1), (X2, Y2)

EXPLANATION

The <Wx, Wy> shows the position at which the graphic cursor is shown first. The graphic cursor is shown by the sign ⌜+⌟, and the color is XORed between the screen color and white.
The value read into the variable is the value of world coordinates.
The <device name> is a character string that specifies the graphic cursor movement and the method of reading. How to specify this is shown below.

Device name	"C" (cursor) or omission	"M" (Mouse)
Graphic cursor moving method	Use the cursor keys	Move the mouse
Timing of reading coordinates	When the return key is pressed	When trigger button A is pressed

When the cursor is moved and the coordinates are read, the specified point ⌜+⌟ remains on the screen. The color of ⌜+⌟ is XORed between the screen color and cyan.
When the coordinates value are all specified "n" places are read and the ⌜+⌟ sign disappears. The maximum number of places at which the coordinate values can be read is 30.

(1) The following explains how to move the graphic cursor using the cursor keys.

1 When a cursor key is pressed, the cursor moves by one dot in the corresponding direction.

2 When a numeric key (0 ~ 9) is pressed, the number of dots for the cursor movement after corresponds to the key pressed. When "0" is pressed, the cursor moves by 10 dots.

3 When a cursor key is pressed while pressing the $\boxed{\text{SHIFT}}$ key, the cursor moves in units of 20 dots. If the $\boxed{\text{SHIFT}}$ key is released, the cursor moves at the same rate as before pressing the key.

4 When the CTRL - K keys or the SHIFT + HOME CLS keys are pressed, the graphic cursor returns to the first position displayed.

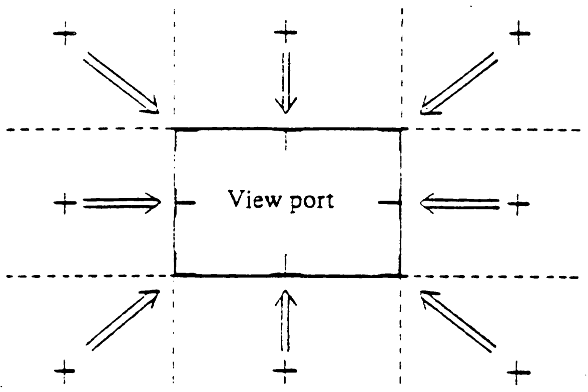
(2) The trigger button moves the mouse as follows when the mouse is used for cursor movement.

1 When trigger button A is pressed, the coordinates value is read.

2 When trigger button B is pressed, the graphic cursor returns to the first position displayed.

NOTE

(1) The graphic cursor never moves out of the view port. If an attempt is made to move the graphic cursor outside of the view port by operating a device (cursor moving key or mouse), the graphic cursor stops. If the <coordinates> is specified as a place outside the view port, the position is set in the view port as shown below.



(2) When the mouse is used, the trigger interrupt of the mouse is set to the interrupt prohibit state. Interrupts become effective again when GCURSOR execution terminates.

REFERENCE

VIEW

SAMPLE PROGRAM

```
LIST

100 'GCURSOR statement
102 CLS
110 GCURSOR(0,0),(A,B),(C,D),(E,F),"M"
120 PRINT"さ゛ひょう (";A;",";B;"),(";C;",";D;"),";
    "(";E;",";F;") か゛ よみこまれました"
130 END

Ready
RUN
さ゛ひょう ( 52 , 26 ),( 52 , 26 ),( 114 , 145 ) か゛ よみこまれました

Ready
```

Line 110 reads the mouse coordinates, and Line 120 outputs these coordinates.

GET@ (Get at mark)

FUNCTION

Reads a graphic pattern or text pattern on the screen into an array.

FORMAT

1 GET (<Sx_1, Sy_1>) – (<Sx_2, Sy_2>), <array name>
2 GET@ (<Sx_1, Sy_1>) – (<Sx_2, Sy_2>), <array name> [, <palette No.>] ...
3 GET@C (<X_1, Y_1>) – (<X_2, Y_2>), <array name>

STATEMENT EXAMPLE GET (10, 10) – (200, 100), A

EXPLANATION

This statement reads a graphic or character pattern displayed in a square having of arbitrary positions <Sx_1, Sy_1> and <Sx_2, Sy_2> as a diagonal into an array.
The reading method of the pattern is specified by the format shown below.

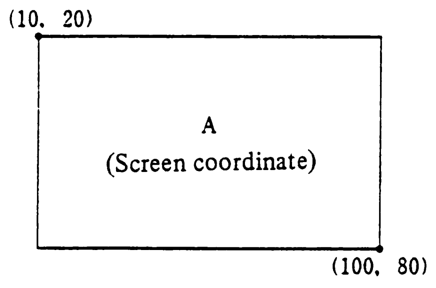
Item		Format 1	Format 2	Format 3
1	Function outline	Reads a graphic pattern on the screen into an array.	Reads a dot pattern of a specified color into an array.	Reads character pattern on the screen into an array.
2	Coordinate system	Screen coordinates	Screen coordinates	Text coordinates
3	Data form in array	Number of dots (lateral) — 2 bytes Number of dots (vertical) — 2 bytes Blue — Number of bytes consisting of [(number of lateral dots + 7) ¥8] number of vertical dots Red Green	Number of dots (lateral) — 2 bytes Number of dots (vertical) — 2 bytes Dot pattern — Number of bytes consisting of [(number of lateral dots + 7) ¥8] number of vertical dots Reads dots of the specified color as "1" and others as "0". R–0: Normal R–1: Reverse I –0: Normal character I –1: IG character SG 00: Displays test only. 01: Displays graphics only. 10: Superimposed with test given priority. 11: Superimposed with graphics given priority.	Number of digits — 1 bytes Number of digits — 1 bytes Character pattern — Number of digits * Number of lines * 2 One character occupies a 2-byte area and the higher byte stores the character attribute, and the lower byte stores the character code. The format is as shown below. ← 1 byte → O | S | I | G | R | Palette No. Character code

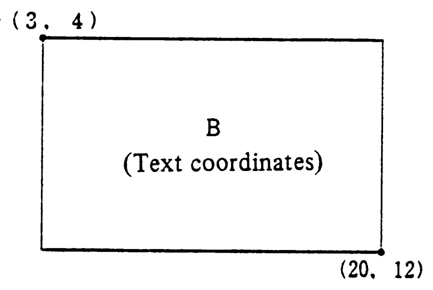
The <max. value of subscript> specified in a DIM statement based on the number of bytes is obtained as shown below.

Integer type	: (Number of bytes + 1) ¥2 − 1
Single-precision type	: (Number of bytes + 3) ¥4 − 1
Double-precision type	: (Number of bytes + 7) ¥8 − 1

(Example)

The following shows the size of array needed in each format.





Format	Coor-dinate	Array size		
		Integer type	Single-precision type	Double-precision type
1	A	2 x 2 + [{(100 − 10 + 1) + 7}¥8] x (80 − 20 + 1) x 3 = 2200		
		(2200 + 1) ¥2 − 1 = 1099 D% (1099)	(2200 + 3) ¥4 − 1 = 549 D ! (549)	(2200 + 7) ¥8 − 1 = 274 D # (274)
2	A	2 x 2 + [{(100 − 10 + 1) + 7 }¥8] x (80 − 20 + 1) = 736		
		(736 + 1) ¥2 − 1 = 367 D% (367)	(736 + 3) ¥4 − 1 = 183 D ! (183)	(736 + 7) ¥8 − 1 = 91 D # (91)
3	B	1 x 2 + (20 − 3 + 1) x (12 − 4 + 1) x 2 = 326		
		(326 + 1) ¥2 − 1 = 162 D% (162)	(326 + 3) ¥4 − 1 = 81 D ! (81)	(326 + 7) ¥8 − 1 = 40 D # (40)

NOTE

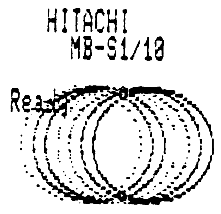
When the description of <Palette No.> in Format (2) is omitted, all dot patterns other than the background color are read into the array.

REFERENCE

PUT @

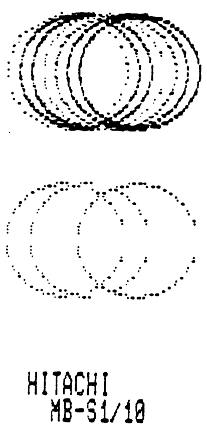
SAMPLE PROGRAM

```
Ready
LIST

100 'GET/PUT statement
110 SCREEN 1: CLS 2: COLOR ,,1
120 DIM A(600),B(200),C(100)
130 FOR I=1 TO 7: CIRCLE (I*8+40,40),40,I:NEXT
140 PRINT "   HITACHI"
150 PRINT "      MB-S1/10"
160 COLOR ,,0
170 ' GET/GET@/GET@C
180 GET     (0,20)-(140,60),A
190 GET@    (0,20)-(140,60),B,1,3,7
200 GET@C   (0,0)-(15,5),C
210 ' PUT/PUT@/PUT@C
220 PUT     (400,10),A,PSET
230 PUT@    (400,60),B,PSET,4
240 PUT@C   (50,15),C
250 END

Ready
```



```
HITACHI
  MB-S1/10
Ready
```

```
LCOPY 4
```



```
HITACHI
  MB-S1/10
```

This program displays the graphic, dot and character patterns read by Lines 180, 190 and 200 at the specified positions on the screen by Lines 220, 230 and 240, respectively.

GOSUB ~ RETURN

FUNCTION	Branches to the subroutine specified by the Line No. and then returns.
FORMAT	GOSUB <Line No.> (The abbreviated form is GOS.) RETURN [<Line No.>] (The abbreviated form is RET.)
STATEMENT EXAMPLE	GOSUB 100

EXPLANATION

Execution is transferred to the subroutine specified by the <Line No.> in the GOSUB statement, and the execution returns to the statement next to the GOSUB statement when the RETURN statement is executed.
When the <Line No.> in the RETURN statement is specified, the execution does not return to the statement next to the GOSUB statement but to the specified line.
A subroutine is separated from the main program and it ends with a RETURN statement. Subroutines can be called as many times as needed by GOSUB statements. One subroutine may contain two or more RETURN statements, but each of them must correctly correspond to a GOSUB statement.
One subroutine can call another subroutine. Multiplexing of subroutines is possible within the limits of memory capacity.

NOTE

In addition to return from a branched subroutine resulting from execution of a GOSUB statement, a RETURN statement is used to return from a branched subroutine resulting from such instructions as ON ~ GOSUB, ON KEY(n) GOSUB, ON COM(n) GOSUB, ON INTERVAL GOSUB, ON MTRIG GOSUB, and ON STRIG GOSUB.
Do not execute a CLEAR statement in a subroutine. If this is done, the execution of a RETURN statement is no longer possible.

REFERENCE

ON GOSUB

SAMPLE PROGRAM

```
Ready
LIST

100 'GOSUB RETURN statement
110 'main routine
120 PRINT "MAIN ROUTINE"
130 PRINT
140 GOSUB 180
150 PRINT "RETURN FROM SUBROUTINE"
160 END
170 '
180 'subroutine
190 FOR I=1 TO 1000:NEXT I
200 PRINT "SUBROUTINE"
210 PRINT
220 FOR I=1 TO 1000:NEXT I
230 RETURN

Ready


RUN
MAIN ROUTINE

SUBROUTINE

RETURN FROM SUBROUTINE

Ready
```

This program shows an example of calling a subroutine from the main routine by a GOSUB statement and returning to the main routine with a RETURN statement.
When lines 110 through 160 are executed, the main routine displays that it has control. Then the control is transferred to the subroutine for Lines 180 to 230 by the GOSUB statement on Line 140. The FOR ~ NEXT statements on Lines 190 and 220 are given to provide a delay so that transfer of the control to the subroutine is displayed. In the subroutine, Line 200 displays that the control is in the subroutine and the RETURN statement on Line 230 returns the control to the statement after the GOSUB statement of the main routine. Line 150 of the main routine shows that the control has been returned to the main routine.

GOTO

FUNCTION	Unconditionally branches to the specified Line No.
FORMAT	GOTO <Line No.> (The abbreviated form is GO.)
STATEMENT EXAMPLE	GOTO 100

EXPLANATION

This statement transfers program control to the specified line number.
When a GOTO statement is executed, the program control unconditionally branches to the line specified by the <Line No.>. If the specified line is a non-execution statement, execution is performed from the first execution statement after the specified line. A non-execution statement means a statement that does not involve direct execution in BASIC such as a REM or DATA statement.

NOTE

A space may be left between "GO" and "TO".

REFERENCE

ON ~ GOTO

SAMPLE PROGRAM

```
LIST

10 'GOTO statement
20 GOTO 40
30 I=1:PRINT"line 30  I=";I:END
40 PRINT"line 40  I=";I:GOTO 30

Ready
RUN
line 40   I= 0
line 30   I= 1

Ready
```

This program shows an example of using GOTO statements in lines 20 and 40. The order of printing is in reverse to the list.

I

IF ~ THEN ~ ELSE

FUNCTION

Branches depending on the value of a logical expression.

FORMAT

```
1  IF <logical expression> THEN
    | <statement> [ : <statement> ] ... |
    |           <Line No.>              |
                 | <statement> [ : <statement> ] ... |
   [ELSE         |           <Line No.>              |  ]
2  IF <logical expression> GOTO <Line No.>
                 | <statement> [ : <statement> ] ... |
   [ELSE         |           <Line No.>              |  ]
```

STATEMENT EXAMPLE

```
(1)  IF A = B THEN PRINT "EQVAL" ELSE 100
(2)  IF A$ = "ABC" GOTO 100 ELSE 200
```

EXPLANATION

Program execution is controlled by the condition of logical expression.
If the condition described in the <logical expression> is true (that is, other than "0"), the description after THEN or the GOTO statement is executed. Either a <Line No.> or execution statement can be written in the THEN statement. Only a <Line No.> can be written after GOTO. This lines number shows the branch destination of the program. If the condition described in the <logical expression> is not true (that, "0"), the description after THEN or the GOTO statement is ignored, and if there is a description started by ELSE, the description contents are executed. If there is no description started by ELSE, the execution goes to the next line.
In IF ~ THEN ~ ELSE statements, a different IF statement can be written after THEN or ELSE for multiplexing. In this case, no parts other than the last ELSE can be omitted. Also, multiplexing is possible within the limit of one line (not exceeding 255 characters).

SAMPLE PROGRAM

```
LIST

100 'IF THEN ELSE statement
110 INPUT"ENTER TWO VALUE,A and B";A,B
120 IF A=B THEN PRINT "A is as same as B.":GOTO 140
130 IF A<B THEN PRINT "A is smaller."
    ELSE PRINT "A is larger."
140 GOTO 110

Ready
RUN
ENTER TWO VALUE,A and B? 1,2
A is smaller.
ENTER TWO VALUE,A and B? 2,2
A is as same as B.
ENTER TWO VALUE,A and B? 3,2
A is larger.
ENTER TWO VALUE,A and B?

Break In 110
Ready
```

This program inputs two numeric values of A and B and determines which is greater.
The INPUT statement on Line 110 inputs the A and B values. The IF statement on Line 120 checks whether or not A is equal to B. If the two are equal, the effect is notified and the program returns to Line 110. If they are not equal, the execution goes to Line 130 and whether or not A is smaller than B is checked. If A is smaller than B, this is notified. If A is not smaller than B, the fact that A is greater than B is notified because of the description after ELSE. When the comparison is complete, the program goes back to Line 110 and the computer waits for an input. Refer to the execution result.

IGS

FUNCTION

Sets the IG character pattern by definition of data for an IG character.

FORMAT

```
IG$ (<character code>) = ["<character string 1>"]
[,["<character string 2>"] [,"<character string 3>"]]
```

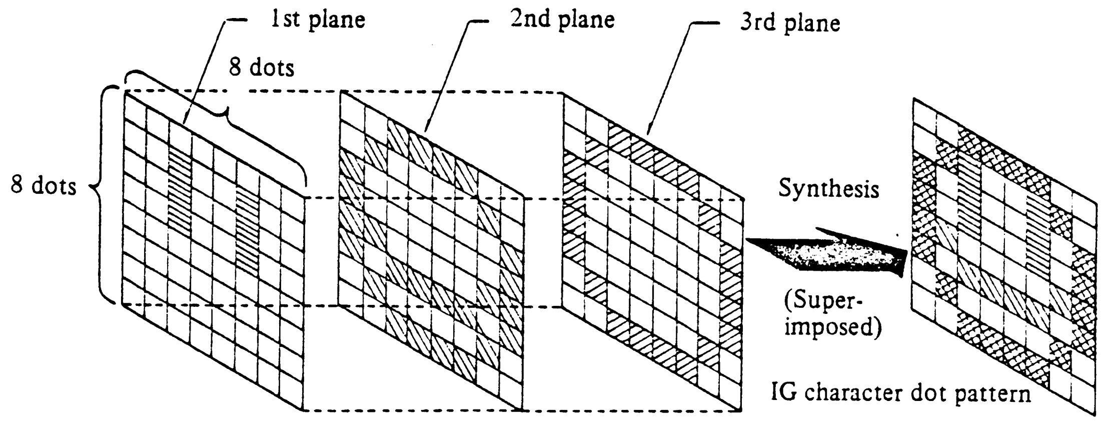
EXPLANATION

The <character code> is the code that corresponds to the IG character. Specify a value in the range from "0" to "255".

Specifying this <character code> in a BASIC PRINT statement causes the defined IG character to be displayed on the screen. In this case, set the IG switch of the COLOR statement to ON.

The size of one character pattern that can be made by IG is the same as the size of one character shown in the character code table, and it is 8 dots in each the vertical and horizontal directions. IG can control each dot in this character and each dot can be colored in one of a maximum of eight colors.

In order to color each dot, three planes are synthesized. Each dot resulting from the synthesis of three planes makes a Palette No. and the color can be instantaneously changed by a PALETTE statement.



The correspondence between set (1) and reset (0) and Palette No. of dots on each plane is as shown below.

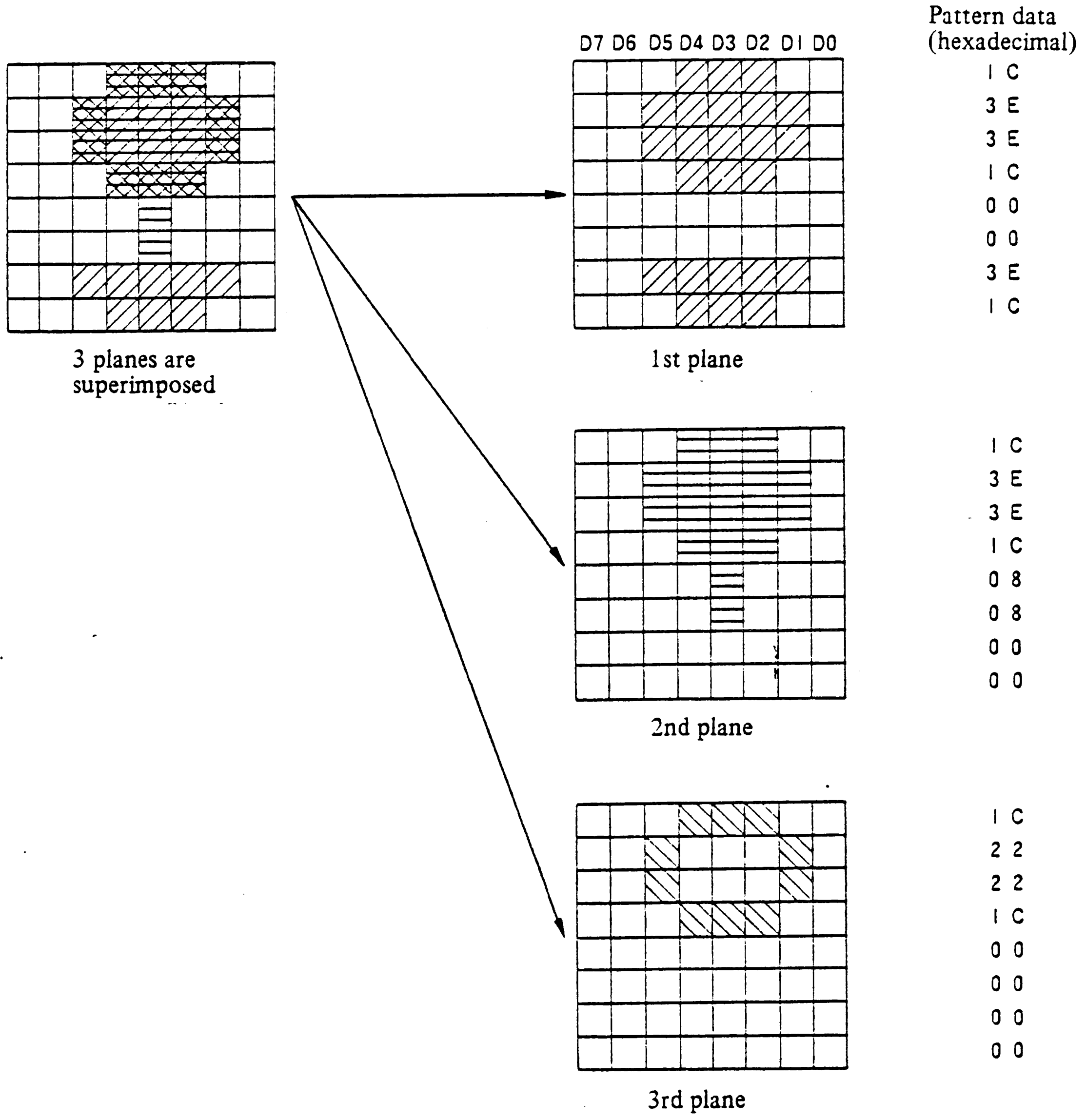
Palette No.	Initial color of palette	1st plane (red)	2nd plane (green)	3rd plane (blue)
0	Black (Nothing is displayed)	0	0	0
1	Blue	0	0	1
2	Red	1	0	0
3	Magenta (purple)	1	0	1
4	Green	0	1	0
5	Cyan (aquamarine)	0	1	1
6	Yellow	1	1	0
7	White	1	1	1

Each of <character string 1>, <character string 2> and <character string 3> is a hexadecimal character string that defines the dot pattern of the first, second and third planes, respectively. The hexadecimal character determines which dots are to be set and which dots are to be reset, and the dots that correspond to "1" as expressed in binary notation are set and the dots that correspond to "0" are reset.

The hexadecimal character strings are defined in the sequence of patterns from the upper character to the lower character. If a character string of one plane is longer than the length of one character (8 x 2 characters), the first 16 characters remain as effective characters and the rest are lost. When a character string is shorter than the length of one character, when the length is an odd number, or when it contains a character other than hexadecimal (0 ~ 9, A ~ F), the character(s) after the particular characters are ignored, and "00" is set.

However, when the specification of the character string is omitted, the character string length is "0", or the first character is a character other than hexadecimal, the plane is not defined and the data of preceding plane is used.

An example of using IG$ is shown below. (In order to make it easily understandable, the palette is set to the initial state.)



In an actual program, define as shown below:

$\text{IG\$ (\&H50)="} \underbrace{\text{1C3E3E1C00003E1C}}_{\text{Pattern data of 1st plane}} \text{","} \underbrace{\text{1C3E3E1C08080000}}_{\text{Pattern data of 2nd plane}} \text{","} \underbrace{\text{1C22221C00000000}}_{\text{Pattern data of 3rd plane}} \text{"}$

The pattern data of the third plane can also be defined as "1C22221C".

NOTE

(1) Although the IG data can be defined by specifying a numeric in the range from "0" to "255" for the <character code>, the <character code> that can be displayed on the screen by a PRINT statement is in the range of 32 ~ 126 and 128 ~ 255.
A PUT@C statement must be used to display the IG data for <character codes> "0" through "31" and "127".

(2) During execution of a program in the IG mode, if execution is temporarily stopped by keying in "Break", the screen mode of text returns to the CG mode from the IG mode. However, if "COLOR,,1" is input, the processing is continued in the IG mode.

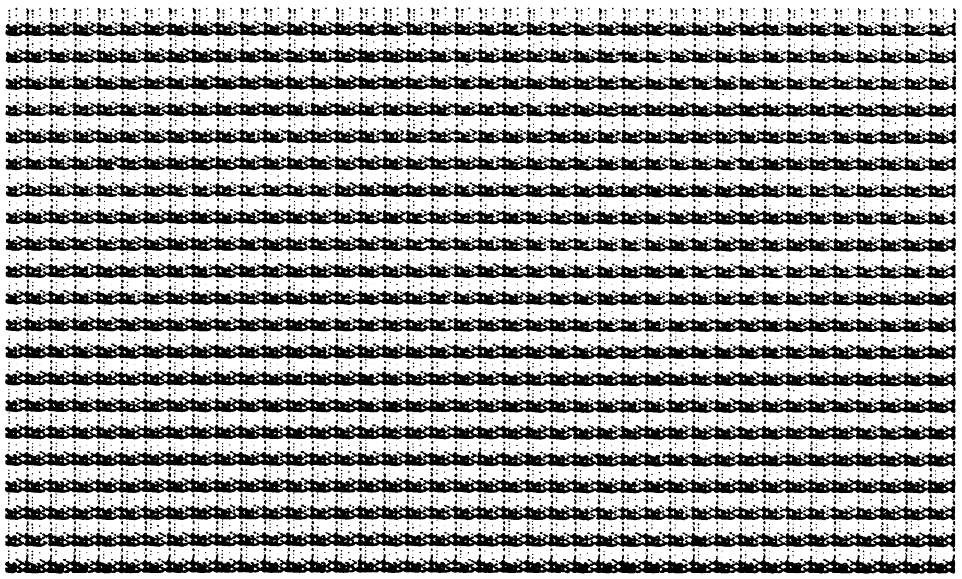
(3) When a character is set to black by IGS, the color cannot be changed by PALETTE.

REFERENCE

IMAGE

SAMPLE PROGRAM

```
Ready
LIST

100 'IG$ statement
110 CLS:SCREEN 1:WIDTH 80
120 COLOR ,,1
130 IG$(130)="COCOCOCOCOCOCOCO","3030303030303030",
    "0303030303030303"
140 IG$(131)="0303030303030303"
150 FOR X=1 TO 960
160 PRINT CHR$(130);CHR$(131);
170 NEXT
180 COLOR ,,0
190 END

Ready
```



```
Ready
LCOPY 4
```

Line 120 sets the IG switch to ON, Lines 130 and 140 define the character pattern, and Lines 150 to 170 display them on the screen.

IMAGE

FUNCTION

Reads the dot pattern from the graphic screen and sets the pattern of an IG character.

FORMAT

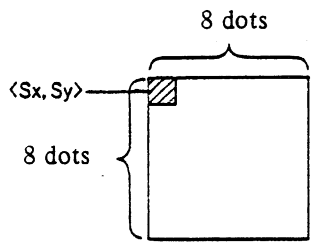
IMAGE (<Sx, Sy>), <character code>

STATEMENT EXAMPLE

IMAGE (50, 50), 60

EXPLANATION

The <Sx, Sy> is the position at which reading of the dots from the graphic screen starts. Specify the dot at the upper left using the screen coordinates. The read range is eight dots each in the vertical and lateral directions.



The <character code> shows the character code that corresponds to the IG character. Specify it in the range 0 ~ 255.

NOTE

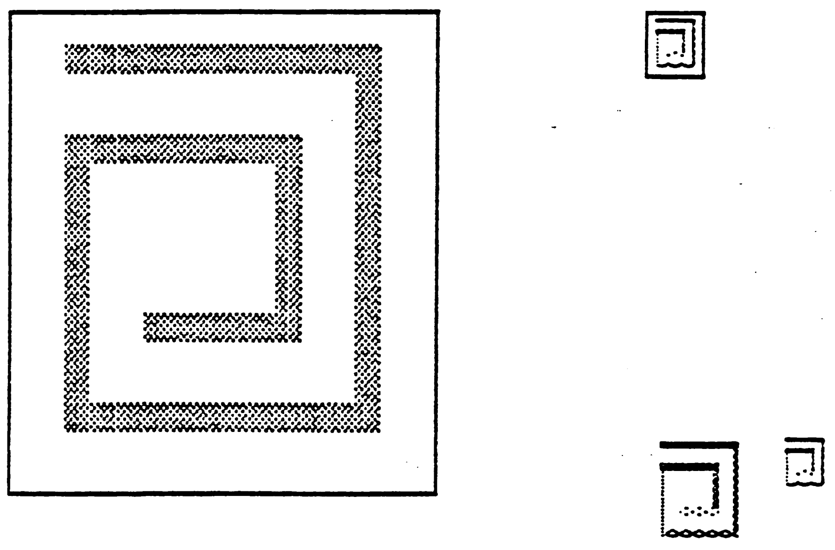
The IG data can be set by specifying the <character code> in the range 0 ~ 255, but the <character codes> that can be displayed on the screen by a PRINT statement are limited to those in a range of 32 ~ 126 and 128 ~ 255.
A PUT@C statement must be used to display the IG data of the <character code> in a range of 0 ~ 31 and 127.

REFERENCE

IG$

SAMPLE PROGRAM

```
10 WIDTH 40:SCREEN 0:LINE(14,7)-(272,136),,,B
20 WINDOW(0,0)-(15,15):VIEW(200,8)-(215,23),,7:LOCATE 1,1
30 A=INSTR("01234567*+-"+MKI$(&H1C1D)+MKI$(&H1E1F)+CHR$(13),INPUT$(1))
40 ON A GOSUB 160,160,160,160,160,160,160,160,130,140,150,60,70,80,90,170
50 GOTO 30
60 SX=X+1 AND &HOF:GOTO 100
70 SX=X-1 AND &HOF:GOTO 100
80 SY=Y-1 AND &HOF:GOTO 100
90 SY=Y+1 AND &HOF
100 ON MD GOSUB 110,120:X=SX:Y=SY:LOCATE X+1,Y+1:RETURN
110 PSET(X,Y):PRINT CHR$(254);:RETURN
120 PRESET(X,Y):PRINT " ";:RETURN
130 MD=0:RETURN
140 MD=1:RETURN
150 MD=2:RETURN
160 COLOR A-1:RETURN
170 COLOR 7,0:LOCATE 0,20
180 INPUT"IG Char code = ";IG:COLOR 7,0,1
190 IMAGE (0,0),IG:IMAGE(8,0),IG+1:IMAGE(0,8),IG+2:IMAGE(8,8),IG+3
200 WINDOW(0,0)-(639,199):VIEW(0,0)-(319,199):LOCATE 30,15
210 PRINT CHR$(IG)+CHR$(IG+1)+MKI$(&H1D1D)+CHR$(&H1F)+CHR$(IG+2)+CHR$(IG+3)
220 SYMBOL(200,120),CHR$(IG+0)+CHR$(IG+1),2,2
230 SYMBOL(200,136),CHR$(IG+2)+CHR$(IG+3),2,2
240 COLOR 7,0:LOCATE 0,21:END
```



```
IG Char code = ? &H80
Ready
```

This program defines an IG pattern while moving the cursor. Pressing the $\boxed{+}$ key sets the mode to set dots while moving the cursor, and pressing the $\boxed{-}$ key sets the mode to reset dots while moving the cursor. Pressing the $\boxed{*}$ key sets the initial mode of moving the cursor. Specify the color of dots to be set using a numeric key ranging from $\boxed{0}$ to $\boxed{7}$. Pressing the $\boxed{\hookleftarrow}$ key terminates the definition of the pattern, and the computer asks for the IG character code.

The pattern that can be defined is a 16 x 16 dot pattern consisting of four IG characters where character codes are used continuously. Since the defined pattern is displayed in the frame at the upper right as a graphic pattern, the IMAGE statement on Line 190 reads it and Lines 210 through 230 display it using a PRINT statement and SYMBOL statement.

INPUT

FUNCTION	Inputs from the keyboard.
FORMAT	INPUT ["<prompt statement>" | ; , |] <variable name> [,<variable name> ...]
STATEMENT EXAMPLE	INPUT 'NAME" ;A$, B$

EXPLANATION

This statement reads input from the keyboard and assigns it to a variable.
When an INPUT statement is executed, the program execution stops, a question mark ⌜?⌟ is output, and the computer waits for input from the keyboard. If the <prompt statement> is given, the statement is output before the question mark. When the <prompt statement> is given and a comma is used after it as a delimiter, the question mark is not output.
Data input is complete when the $\boxed{\lrcorner}$ key is pressed and the data is assigned to a variable or variables. The number of input data items and their form must coincide with those of the variables. Use a comma ⌜,⌟ or colon ⌜:⌟ when inputting two or more data items to delimit them. A Hollerith constant input needs not be enclosed by quotation marks, but if a comma ⌜,⌟ or colon ⌜:⌟ is used as a part of data, the character string must be enclosed by double quotation marks ⌜"⌟. If the number of input data items and the form do not concur with the variables, or when the $\boxed{\lrcorner}$ key is pressed without inputting any data for numeric variables, the computer outputs ⌜? Redo From Start⌟ and it waits for another input. Note that an ON ERROR GOTO statement cannot handle error processing.
When execution of a program is temporarily stopped for an INPUT statement, execution restarts from the INPUT statement when a CONT statement is input.

NOTE

This statement cannot be used as a direct command.

REFERENCE

LINE INPUT, INKEY$ function, INPUT WAIT, INPUT$ function

SAMPLE PROGRAM

```
Ready
LIST

100 'INPUT statement
110 OPEN "O",#1,"ADDRESS"
120 INPUT "ENTER YOUR ZIP CODE";ZIP$
130 INPUT"ENTER YOUR ADDRESS";ADDR$
140 INPUT"ENTER YOUR TELEPHONE NUMBER";TEL$
150 INPUT"ENTER YOUR NAME";NAM$
160 PRINT "THANK YOU! I'LL TELEPHONE YOU."
170 PRINT #1,ZIP$,ADDR$,TEL$,NAM$
180 CLOSE #1
190 END

Ready
RUN
ENTER YOUR ZIP CODE? 123
ENTER YOUR ADDRESS? TOKYO
ENTER YOUR TELEPHONE NUMBER? 110
ENTER YOUR NAME? HITACHI
THANK YOU! I'LL TELEPHONE YOU.

Ready
```

This is an example of a program to make an address file using INPUT statements.
Since an INPUT statement outputs a message using its <prompt statement> when it is input, it is very convenient to inform the operator what is to be input.

INPUT WAIT

FUNCTION

This is an input from the keyboard with a time limit.

FORMAT

INPUT WAIT <Line No.>;<waiting time>, ["<prompt statement>" | ; , |] <variable name> [,<variable name>] ...

STATEMENT EXAMPLE

INPUT WAIT 50 ; 10, "つぎのては" ;X, Y

EXPLANATION

This statement sets a time limit for input.
The computer waits for the input of an INPUT statement for the number of seconds specified in the <waiting time>. If the input is not completed within the set time, execution goes to the line specified in the <Line No.>. The <waiting time> is in seconds ranging from "1" to "255".
The function of this statement is the same as an INPUT statement, except here a time limit is given.

NOTE

INPUT WAIT cannot be used as a direct command.
If this statement is followed by multi-statements, and input is not conducted within the time limit, the succeeding statements are ignored and the execution goes to the line of <Line No.>.

REFERENCE

INPUT

SAMPLE PROGRAM

```
Ready
LIST

100 ' INPUT WAIT statement
110 INPUT WAIT 160;10,"あなたの なまえは";A$
120 PRINT "あなたの なまえは  ";A$
130 INPUT WAIT 160;5,"あなたの ねんれいは";B$
140 PRINT "あなたの ねんれいは  ";B$
150 GOTO 110
160 PRINT "TIME OVER"
170 END

Ready
RUN
あなたの なまえは? S1
あなたの なまえは  S1
あなたの ねんれいは?
TIME OVER

Ready
```

In this program, if key input is not completed within 10 seconds on Line 110 and 5 seconds on Line 130, "TIME OVER" (Line 160) is displayed and the program terminates.

INPUT

FUNCTION	Reads data from an input file.
FORMAT	INPUT # <File No.>, <variable name> [,<variable name>] ...
STATEMENT EXAMPLE	INPUT # 5, A, B

EXPLANATION

This statement inputs data from a specified file. The specified file must be opened in advance in input mode "I".
「,」, 「:」 and line feed code are used as data delimiters and cannot be used in the data itself.
The term "line feed code" means the character of character code 13 (&H0D), or continuation of a character having character code 13 (&H0D) and a character having character code 10 (&H0A).
If the file is a cassette file, the only line feed code is used as a data delimiter, and 「,」 and 「:」 can be used in the data itself.

NOTE

INPUT # cannot be used as a direct command.

REFERENCE

OPEN, CLOSE

SAMPLE PROGRAM

```
Ready
LIST

100 'INPUT# statement
110 OPEN "I",#1,"ADDRESS"
120 INPUT#1,ZIP$,ADDR$,TEL$,NAM$
130 PRINT "ZIP CODE",ZIP$
140 PRINT "ADDRRESS",ADDR$
150 PRINT "TELEPHONE NUMBER",TEL$
160 PRINT "NAME",NAM$
170 CLOSE #1
180 END

Ready
RUN
ZIP CODE        123
ADDRRESS        TOKYO
TELEPHONE NUMBER              110
NAME            HITACHI

Ready
```

This shows an example of reading a sequential file using an INPUT # statement.
The sequential file "ADDRESS" is the address file prepared by the INPUT statement example program.
Line 110 opens this file.
Since an INPUT # statement is the same as an INPUT statement except that it reads data from a sequential file, the number and form of the variables must concur with the data to be read.
Line 120 reads the contents of the sequential file into four sequential character variables.
Line 130 and succeeding lines output the read character strings.

INTERVAL

FUNCTION — Specifies the time interval of an interval timer interrupt.

FORMAT — INTERVAL <interval>

STATEMENT EXAMPLE — INTERVAL 500

EXPLANATION

The <interval> shows the time interval for an interrupt. Specify it in second units (integer). The <interval> can be specified for a length of up to 32767 seconds.
When the computer is set to the interrupt enable state by an INTERVAL ON statement and an INTERVAL statement is executed, an interrupt is generated at the specified interval in second units and control is transferred to the interrupt processing routine defined by an ON INTERVAL GOSUB statement.
Counting of the time is conducted immediately after executing an INTERVAL ON statement, immediately after executing an INTERVAL statement and immediately after an interrupt generation, and the next interrupt occurs upon reaching the specified interval in the second units.

REFERENCE

INTERVAL ON/OFF/STOP, ON INTERVAL GOSUB

SAMPLE PROGRAM

```
Ready
LIST

100 ' INTERVAL statement
110 SCREEN 1,0,0
120 A=0
130 ON INTERVAL GOSUB 170
140 INTERVAL 5
142 INTERVAL ON
150 IF A<>5 THEN GOTO 150
160 INTERVAL OFF:END
170 A=A+1:PRINT"INTERVAL";A;"かいめ"
180 RETURN

Ready
RUN
INTERVAL 1 かいめ
INTERVAL 2 かいめ
INTERVAL 3 かいめ
INTERVAL 4 かいめ
INTERVAL 5 かいめ

Ready
```

Line 142 of this program permits an interrupt and interrupt processing is applied to the line defined by Line 130 when the time (seconds) defined by Line 140 elapses.

INTERVAL ON/OFF/STOP

FUNCTION	Permits, inhibits or stops interval timer interrupts.
FORMAT	INTERVAL { ON / OFF / STOP }
STATEMENT EXAMPLE	INTERVAL ON

EXPLANATION

INTERVAL ON enables an interrupt at an interval specified by an INTERVAL statement.
INTERVAL OFF inhibits interrupts.
INTERVAL STOP temporarily stops interrupts. When this statement is executed, an interrupt is accepted but the actual interrupt branch to the line specified by an ON INTERVAL GOSUB statement is temporarily suspended. The interrupt action starts when an INTERVAL ON statement is executed.
An INTERVAL STOP statement is effective in the INTERVAL ON state.

NOTE

Execute an INTERVAL OFF statement when terminating a program. The INTERVAL OFF state can be obtained by executing CLEAR, NEW or RUN.

REFERENCE

INTERVAL, ON INTERVAL GOSUB

SAMPLE PROGRAM

Refer to the SAMPLE PROGRAM shown for INTERVAL.

KANJI

FUNCTION

Displays a Kanji at a specified position on the screen.

FORMAT

1 KANJI [(<Sx, Sy>),] <Kanji code> [,<Kanji code>] ...

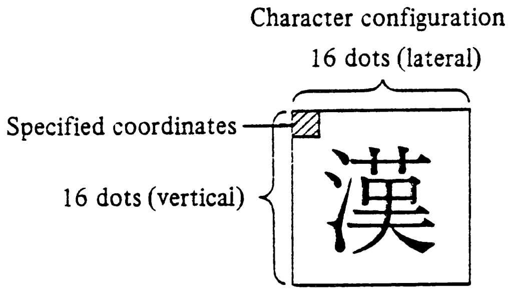
2 KANJI@ [[(<Sx, Sy>)] [,[<plot option>] [,[<color>] [,[<horizontal magnification>] [,<vertical magnification>]]]] ;] <Kanji code> [,<Kanji code>] ...

STATEMENT EXAMPLE

```
1 KANJI (100, 50), &H3021
2 KANJI@ (50, 10), PSET, 5, 2, 3; &H3021
```

EXPLANATION

<Sx, Sy> is the position at which the display of the Kanji starts. Specify the dot at the upper left of the required display area using screen coordinates. Make sure that the coordinates allow the Kanji is contained in the view port without missing any part of it.



When the specification of the Kanji display position is omitted, the Kanji is displayed at the position next to the position specified by a preceeding KANJI statement.
If, however, the specification of the Kanji display position is omitted the first time, the display starts at the reference position (0, 0). If the position specified by the preceeding KANJI statement is at the lower right corner (X = 625, Y = 185) of the screen, an error occurs.
Specify the <Kanji code> using the JIS code. Use decimal or hexadecimal notation (Example:.「亜」= &H3021).
When two or more Kanji codes are specified in succession, Kanji characters are displayed continuously running in the horizontal direction. Kanji characters that cannot be displayed in the current horizontal line are displayed on the next line.

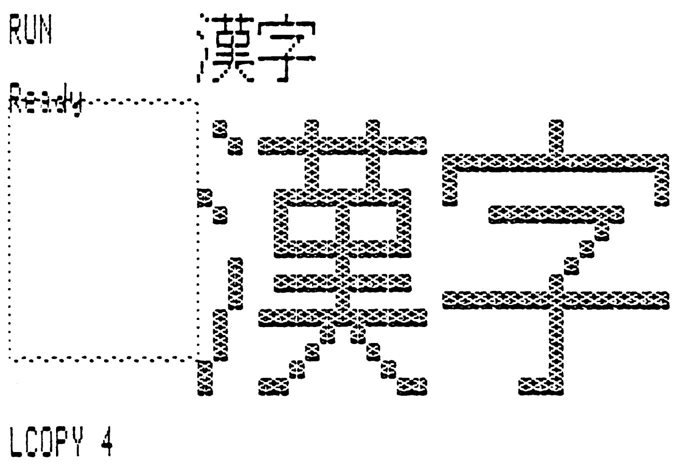
PSET, PRESET, AND, OR, XOR or NOT can be specified for the <plot option>. For further explanation, refer to FORMAT 2 of the PUT@ statement.
The <color> shows the color of the character to be displayed, and when this specification is omitted, the character color of a COLOR statement executed immediately before is taken.
The <horizontal magnification> and <vertical magnification> are the magnification of the Kanji character to be displayed in the horizontal and vertical directions expressed by integers, and the Kanji is displayed on the screen with the size magnification x 16 dots. When the magnification is not specified, "1" is assumed. For further details, refer to the explanation of the PUT@ statement.

REFERENCE

SYMBOL

SAMPLE PROGRAM

```
LIST

100 'KANJI Statement
110 SCREEN 0 : KANJI(50,0),&H3441,&H3B7A
120 LINE(0,20)-(100,80),PSET,1,B
130 KANJI@(50,25),PSET,2,4,4;&H3441,&H3B7A
140 END

Ready
```



```
RUN         漢字
Ready
            漢字
LCOPY 4
```

On this program, Line 110 displayes a Kanji character at the specified position (50, 0), and Line 130 magnifies it to a size four times the ordinary size.

KEY

FUNCTION	Defines the meanings of programmable function keys.
FORMAT	KEY <PF key No.>, <character expression>
STATEMENT EXAMPLE	KEY 1, "LOAD" + CHR$ (34)

EXPLANATION

This statement assigns a character string to PF keys. Specify a numeral 1 through 10 for the <PF key No.>, that is, specify "1" for PF1. A character string consisting of up to 15 characters can be defined for each key.
Define the $\boxed{\lrcorner}$ key (character code 13) and 「"」 (character code 34) using a CHR$ function.
When the power switch is turned off or when a NEW ON command is executed, the PF keys are set back to their initial significance.

REFERENCE

KEY LIST, KEY (n) ON/OFF/STOP, ON KEY (n) GOSUB

SAMPLE PROGRAM

```
LIST

100 'KEY command
110 KEY 10,"AUTO 100"+CHR$(13)
120 KEY LIST
130 NEW

Ready
RUN

PF1     LOAD
PF2     ?DATE$,TIME$
PF3     KEY
PF4     LIST
PF5     RUN
PF6     TERM
PF7     SCREEN
PF8     COLOR
PF9     LIST"LPTO:"
PF10    AUTO 100

Ready
```

This program is used to define a function key prior to preparing a program. The KEY command described on Line 110 defines a character string and control code to a function key. In this case, "AUTO 100" and the carriage return code are defined. Line 120 displays the function key details on the screen. The NEW command on Line 130 erases the current program so that the next program can be prepared.

KEY LIST

FUNCTION — Displays the definition of the programmable function keys.

FORMAT — KEY LIST

STATEMENT EXAMPLE — KEY LIST

EXPLANATION

This statement displays the character string defined to each programmable function key on the screen, while showing the key Nos.
BASIC automatically assigns a character string to each programmable function key at the time of turning on the power switch or when a NEW ON command is executed. The execution result in this state is shown below.

```
KEY LIST

PF1     LOAD CR
PF2     ?DATE$,TIME$CR
PF3     KEY
PF4     LIST CR
PF5     RUN CR
PF6     TERM
PF7     SCREEN
PF8     COLOR
PF9     LIST"LPT0:"CR
PF10    CONT CR

Ready
LCOPY3
```

REFERENCE

KEY

KEY(n) ON/OFF/STOP

FUNCTION	Permits, inhibits or stops programmable function key interrupts.
FORMAT	1 KEY (<PF key No.>) ON 2 KEY (<PF key No.>) OFF 3 KEY (<PF key No.>) STOP
STATEMENT EXAMPLE	KEY (1) ON

EXPLANATION

As an easy explanation, "n" (an integer in the range of 1 ~ 10) is used for the <PF key No.>. When Format 1 is executed, the PF key to which "n" is specified as the <PF key No.> (PFn), has the function as an interrupt key, and it temporarily loses its function as a user definable key.
When PFn is pressed in this state, the subroutine specified by an ON KEY(n) GOSUB statement (interrupt processing routine) is performed. If no ON KEY(n) GOSUB statement has been executed, the interrupt is not executed immediately even if PFn is pressed, it is executed when an ON KEY(n) GOSUB statement is reached in the program. In either case, the PFn key cannot function as a user definable key. This state is released when Format 2 or an instruction like CLEAR, RUN, or NEW is executed.
When Format 2 is executed, PFn loses the function of being an interrupt key and its function of being a user definable key is restored. However, since the ON KEY(n) GOSUB statement is not released, if Format 1 is executed again, the preceding interrupt processing routine can be executed by pressing PFn.
When Format 3 is executed, no interrupt processing routine is executed even when PFn is pressed. However, the computer stores the fact that PFn was pressed, and the interrupt processing routine is executed when Format 1 is executed. However, since the execution of KEY(n) STOP is in the KEY(n) OFF state, Format 3 must be executed in the KEY(n) ON state.
All programmable function keys are set to the KEY(n) OFF state when an instruction like RUN, NEW or CLEAR is executed.

REFERENCE

KEY, ON KEY(n) GOSUB

SAMPLE PROGRAM

Refer to the "PF key interrupt control" sample program in the APPENDIX.

LCOPY

FUNCTION Outputs the contents of the screen display on a printer.

FORMAT LCOPY [<Function No.>]

STATEMENT EXAMPLE LCOPY 0

EXPLANATION

The <Function No.> specifies the screen display to be output to a printer.

Function No.	Contents
0 or no specification	Outputs a text screen and graphic screen in overlapped state.
1	Outputs a graphic screen only.
2	Outputs a text screen as it is.
3	Outputs a text screen as graphics.
4	Overlaps a text screen and graphic screen and outputs the result using tiling. (Note 1)
5	Outputs a graphic screen using tiling.
6	Outputs a text screen using tiling. (Note 2)
7	The same as 6.

(Note 1): Tiling: Each color is changed to an different pattern and is output to the printer.
(Note 2): This is effective when outputting an IG pattern.

An LCOPY statement can be used for output to one of the following printers only.

(○: Possible, ×: Not possible)

Function No.	0 or no specification	1	2	3	4	5	6	7
MP-1041 MP-1050	○	○	○	○	○	○	○	○
MP-1020 MP-1052	×	×	○	×	×	×	×	×
MP-1053	×	×	○	×	×	×	×	×

(Note) Since Kanji is displayed on a graphic screen, it can be printed by the MP-1041 or MP-1050.

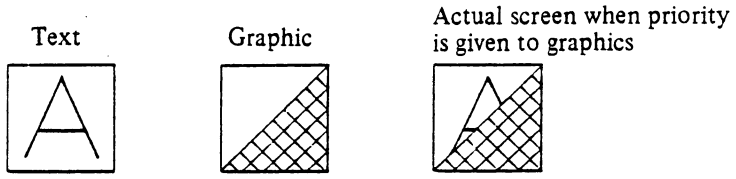
A hard copy can be obtained by pressing the following keys.

$\boxed{\text{COPY}}$	=LCOPY 0
$\boxed{\text{COPY}}+\boxed{\text{GRAPH}}$	=LOCOY 1
$\boxed{\text{COPY}}+\boxed{\text{SHIFT}}$	=LCOPY 2
$\boxed{\text{COPY}}+\boxed{\text{GRAPH}}+\boxed{\text{SHIFT}}$	=LCOPY 3
$\boxed{\text{COPY}}+\boxed{\text{CTRL}}$	=LCOPY 4
$\boxed{\text{COPY}}+\boxed{\text{GRAPH}}+\boxed{\text{CTRL}}$	=LCOPY 5
$\boxed{\text{COPY}}+\boxed{\text{SHIFT}}+\boxed{\text{CTRL}}$	=LCOPY 6
$\boxed{\text{COPY}}+\boxed{\text{GRAPH}}+\boxed{\text{SHIFT}}+\boxed{\text{CTRL}}$	=LCOPY 7

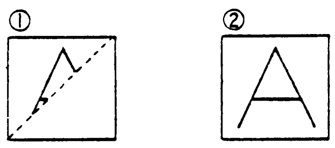
NOTE

(1) When executing LCOPY 2:
IG is output as blank and the reverse mode of CG is output as the ordinary mode.

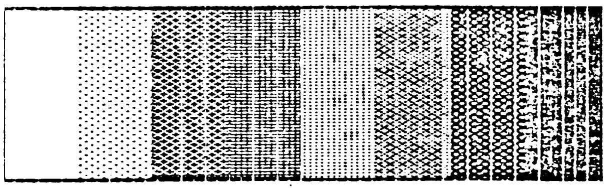
(2) When outputting either a text screen or graphic screen:
(LCOPY 1, 3, 5, 6, 7)
Although the priority of the screen is determined by the attribute, all data stored in the V-RAM is output.
For example, suppose that the screen is as shown below:



If the output of a hard copy (LCOPY 3, 6, 7) of the text screen only is attempted. 2 is output instead of 1 .



(3) The correspondence between each color and tiling pattern when using the tiling function is as shown below.

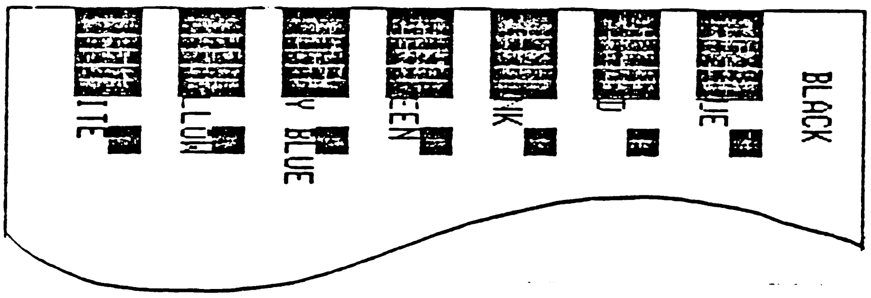


(4) The screen is output to the printer turned through 90° with LCOPY instructions other than LCOPY 2.

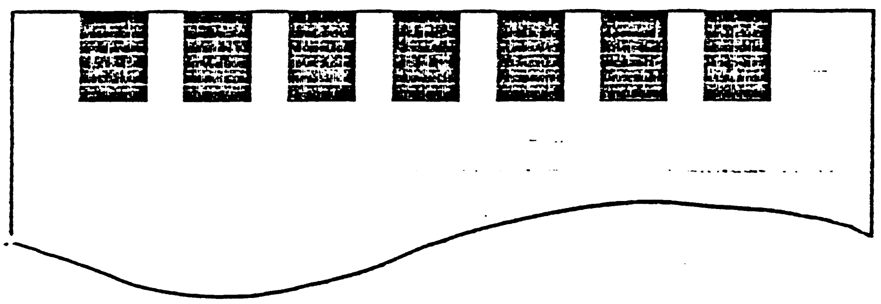
SAMPLE PROGRAM

```
100 ' LCOPY statement
110 SCREEN 1,,0: CLS
120 IG$(&H41)="0000000000000000","0000000000000000","0000000000000000"
130 IG$(&H42)="0000000000000000","0000000000000000","FFFFFFFFFFFFFFFF"
140 IG$(&H43)="FFFFFFFFFFFFFFFF","0000000000000000","0000000000000000"
150 IG$(&H44)="FFFFFFFFFFFFFFFF","0000000000000000","FFFFFFFFFFFFFFFF"
160 IG$(&H45)="0000000000000000","FFFFFFFFFFFFFFFF","0000000000000000"
170 IG$(&H46)="0000000000000000","FFFFFFFFFFFFFFFF","FFFFFFFFFFFFFFFF"
180 IG$(&H47)="FFFFFFFFFFFFFFFF","FFFFFFFFFFFFFFFF","0000000000000000"
190 IG$(&H48)="FFFFFFFFFFFFFFFF","FFFFFFFFFFFFFFFF","FFFFFFFFFFFFFFFF"
200 FOR I=0 TO 7
210   LINE (0,I*24)-(48,15+I*24),PSET,I,BF
220 NEXT
230 COLOR ,,1
240 FOR I=0 TO 7
250   LOCATE 8,I*3: PRINT CHR$(&H41+I);CHR$(&H41+I)
260 NEXT
270 COLOR ,,0
280 LOCATE 4,1: PRINT "BLACK"
290 LOCATE 4,4: PRINT "BLUE"
300 LOCATE 4,7: PRINT "RED"
310 LOCATE 4,10: PRINT "PINK"
320 LOCATE 4,13: PRINT "GREEN"
330 LOCATE 4,16: PRINT "SKY BLUE"
340 LOCATE 4,19: PRINT "YELLOW"
350 LOCATE 4,22: PRINT "WHITE"
360 FOR I=0 TO 7
370   LCOPY I
380 NEXT
390 END
```

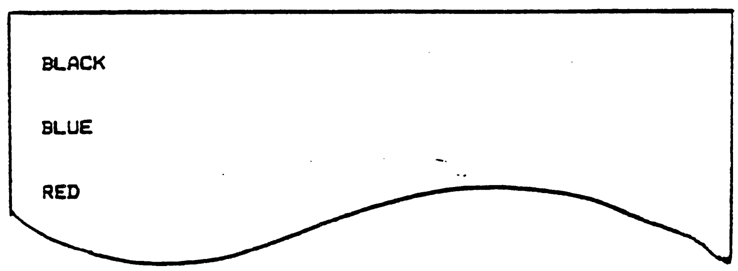
LCOPY 0



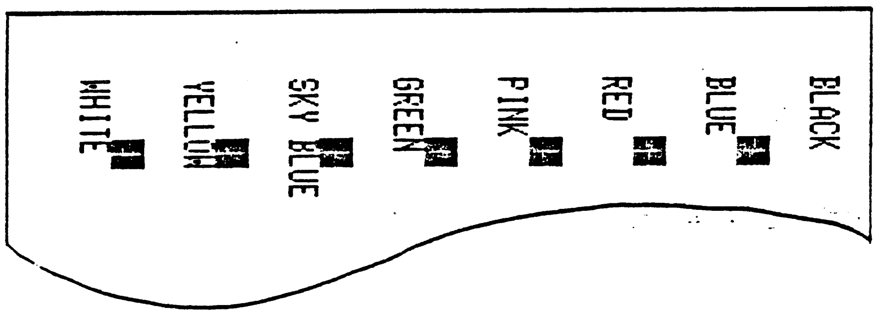
LCOPY 1



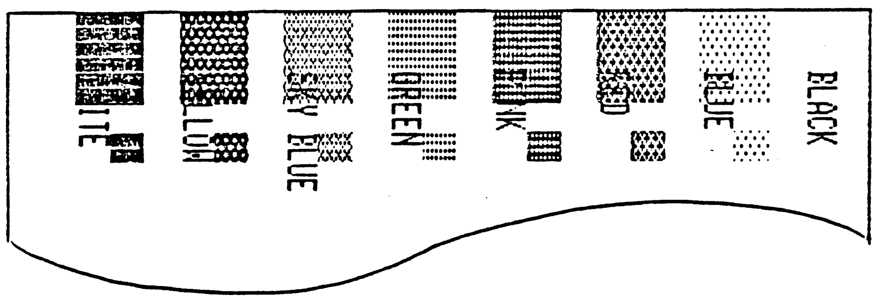
LCOPY 2



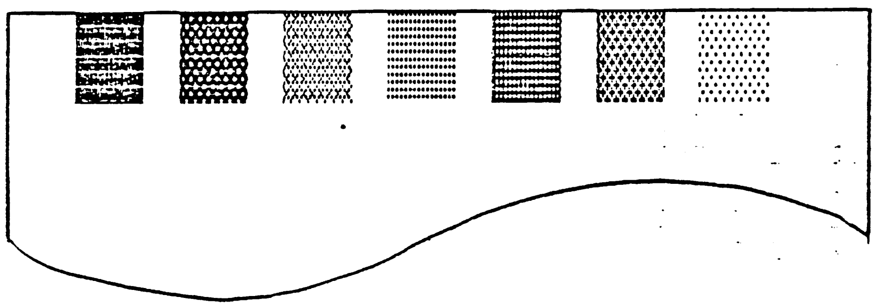
LCOPY 3



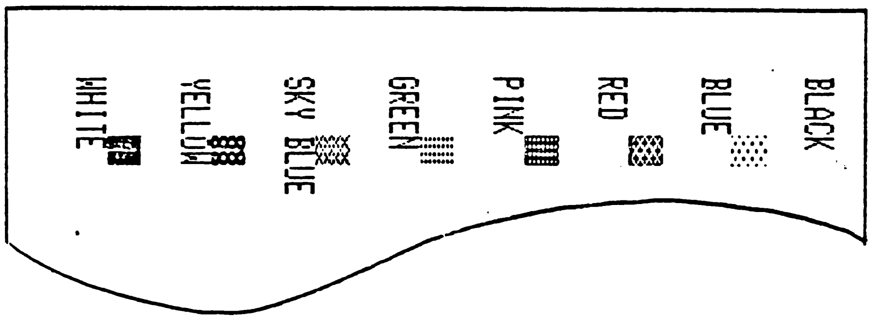
LCOPY 4



LCOPY 5



LCOPY 6
LCOPY 7



Lines 360 ~ 380 of this program copy the screen drawn by Lines 110 ~ 350.

LET

FUNCTION Assigns the value of an expression to a variable.

FORMAT [LET] <variable name> = <expression>

STATEMENT EXAMPLE LET A = B + C

EXPLANATION

This statement assigns the value of <expression> to a variable.
The value of <expression> is assigned to a variable or array. This equals sign means assignment. The word LET may be omitted.
The form of the variable and the form (numeric or character) of the <expression> value must be the same.
A character expression may be specified as the <expression>.

SAMPLE PROGRAM

```
Ready
LIST

100 'LET statement
110 LET A=10
120 LET B=20
130 LET C=30
140     S=A+B+C
150 PRINT S
160 END

Ready
RUN
 60

Ready
```

- Lines 110 through 130 show examples of the use of LET statements. Line 140 shows an example of omitting it.
Line 150 shows the sum of A, B and C.

LFILES

FUNCTION	Outputs file names recorded on a cassette tape to a printer.
FORMAT	LFILES ["CAS0:"]
STATEMENT EXAMPLE	LFILES "CAS0:"

EXPLANATION

This statement inputs file names recorded on a cassette tape after the current position and outputs the result to a printer.
Since a cassette tape has no end mark, the processing does not terminate even the tape is read up to the end.
To terminate processing, press the $\boxed{\text{CTRL}}$ + $\boxed{\text{D}}$ keys.

REFERENCE

FILES

SAMPLE PROGRAM

```
Ready
LIST

100 'LFILES command
110 LPRINT"****** LFILES ******"
120 MOTOR ON
130 LFILES
140 MOTOR OFF
150 END

Ready


RUN


****** LFILES ******

DEFUSR1    0 B
GET-PUT    0 B
PALETTE    0 B
TEST       0 B
```

This program prints the catalog names of program stored on a cassette tape using the printer.

LINE

FUNCTION Draws a line or box on the screen.

FORMAT LINE [(<Wx_1, Wy_1>)] – (<Wx_2, Wy_2>) [,[<plot option>] [,[<color>] [,[| B / BF |] [,<line style>]]]]

STATEMENT EXAMPLE LINE (100, 100) – (200, 100), XOR, 5, BF, &HF99F

EXPLANATION

The specification of <Wx_1, Wy_1> and <Wx_2, Wy_2> draws a line. When the specification of (Wx_1, Wy_1) is omitted, the starting point is LP (refer to 1.10 Coordinate system).

Specify one of PSET, PRESET, AND, OR, XOR, and NOT for the <plot option>. When the specification is omitted, PSET is assumed.

When PSET is specified, the line is drawn in the specified color. When PRESET is specified, the specified color is ignored and the line is drawn in the background color specified by a COLOR statement. Therefore, the part drawn in the background color looks as if the line has not be drawn. When one of AND, OR, or XOR is specified, a logical operation is conducted between the specified color and the color being displayed at present, and the line is drawn in the color determined by the logical operation. Specification of NOT brings about the same result as specifying PRESET.

Specify the color of the straight line as the <color>.

「B」 means a box. When 「B」 is specified, a rectangular frame having the diagonal <Wx_1, Wy_1> to <Wx_2, Wy_2> is drawn. 「BF」 stands for 「Box filled」, and when 「BF」 is specified, inside the frame is filled by lines specified by the <plot option> or <line style>.

When the<line style> is specified, a broken line or line of alternate long and short dashes is drawn depending on the specified value. When it is omitted, a normal straight line is drawn.

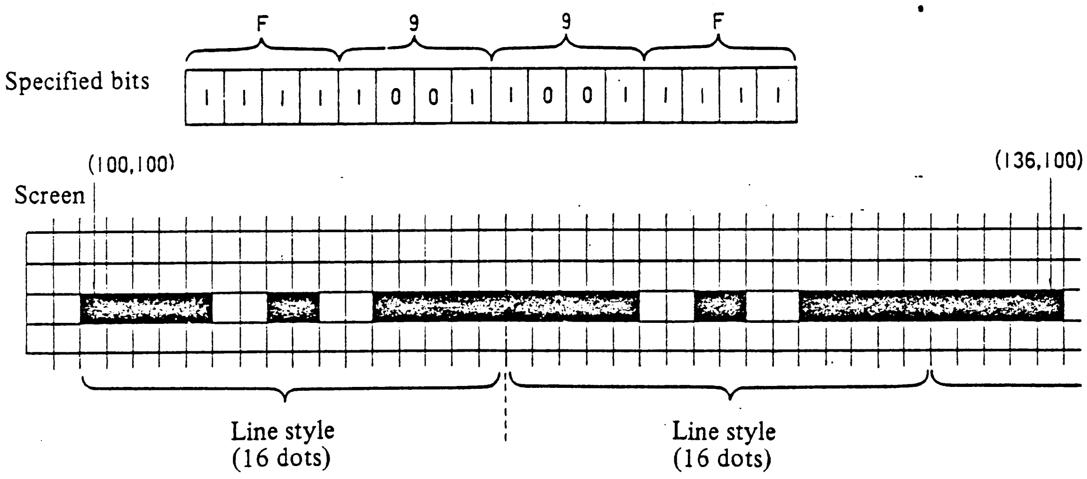
Each of the 16 bits has a 1:1 correspondence with each of the 16 dots on the screen, and the dots that correspond with bits that are set to "1" are displayed and those corresponding to dots that are set to "0" are not displayed. When a line is drawn for a length exceeding 16 dots, this line style is repeated for each of the 16 dots on the screen.

For example, to draw an line of alternate long and short dashes using the line style specification, specify as follows:

```
LINE (100, 100) – (135, 100), PSET, 7,, &HF99F
```

Since the line style is specified as &HF99F, the bits and dots on the screen correspond as shown below.

When <line style> is specified, the specification of <plot option> applies only to dots that correspond to line style bits that are set to "1". Therefore, the line is drawn as follows:



NOTE

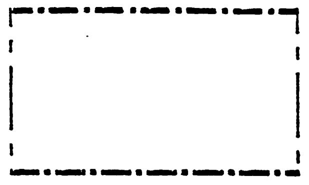
In the 640 x 200 dot mode, even if the same line style is specified, if the gradient is different, the line style looks different. This is because, since the dot intervals in the vertical direction are wider than the dot intervals in the horizontal direction by a factor of about two, the line style changes if the gradient is different. The sharper the gradient, the longer the line style.

REFERENCE

COLOR

SAMPLE PROGRAM

```
LIST

100 'LINE statement
110 LINE(100,100)-(200,125),PSET,4,B,&HF99F
120 END

Ready
```



This program draws a line of alternate long and short dashes because &HF99F is specified in Line 110.

LINE INPUT

FUNCTION	Inputs one whole line without delimiting the contents into a character string.
FORMAT	LINE INPUT ["<prompt statement>" ;/,] <character variable name>
STATEMENT EXAMPLE	LINE INPUT "DATA" ; A$

EXPLANATION

Input character strings as one line (within 255 characters) from the keyboard.
If the <prompt statement> is specified, the computer outputs it at the time of execution and waits for input.
Unlike the INPUT statement, a question mark is not output in this case. The input is complete when data is input and then the ⏎ key is pressed. Any input in excess of 256 characters is ignored.
Colons 「:」 and commas 「,」 can be used.

NOTE

This statement cannot be executed when the computer is waiting for a command.

REFERENCE

INPUT$ function and INKEY$ function

SAMPLE PROGRAM

```
Ready
LIST

100 'LINE INPUT statement
110 PRINT"Can you type following string ?"
120 A$="Personal Computer is very useful."
125 PRINT A$
128 PRINT"Type the string!"
130 LINE INPUT B$
140 IF A$=B$ THEN PRINT"All correct!
   ":GOTO 150 ELSE PRINT"Again!":GOTO 125
150 END

Ready
RUN
Can you type following string ?
Personal Computer is very useful.
Type the string!
Personal Computer is very useful.
All correct!

Ready
```

This is a sample program prepared to practice typing.
Line 120 outputs character strings that should be used as the text, and the same character strings are input by the LINE INPUT statement on Line 130.
If the input character strings match, Line 140 indicates that the typing is correct. If not, a second input is requested.
Since a LINE INPUT statement can read a comma 「,」, whereas an INPUT statement cannot, it is very useful to input English sentences.

LINE INPUT

FUNCTION	Reads one line of data from an input file.
FORMAT	LINE INPUT # <File No.>, <character variable name>
STATEMENT EXAMPLE	LINE INPUT # 1,A$

EXPLANATION

This statement reads one line from the file specified by the <File No.> and assigns the contents to variables. "One line" referred to here means all input items until reading the line feed code (refer to the explanation of 「INPUT #」). However, the number of characters read is limited to 253. The line feed code (carriage return or line feed) cannot be assigned in the character variables.
The file specified by the <File No.> must have been opened in the input mode.
A LINE INPUT # statement is especially useful to input a program that has been saved in the ASCII mode.

REFERENCE

INPUT #

SAMPLE PROGRAM

```
LIST

100 'LINE INPUT# statement
110 OPEN "I",#1,"MESSAGE"
120 IF EOF(1) THEN 160
130 LINE INPUT #1,MESSAGE$
140 PRINT MESSAGE$
150 GOTO 120
160 CLOSE #1
170 END

Ready
RUN
Hello. How are you?
Can you drink with me?
I want to see you so much.
I hope you will come.
Bye-bye.

Ready
```

This program reads and displays the contents of sequential file "MESSAGE".
Line 110 opens the file as "MESSAGE" is input. The IF ~ THEN statements on Line 120 check for the end mark of the file using the EOF function, the LINE INPUT # statement on Line 130 reads the contents into character variable "MESSAGE", and Line 140 outputs the result to the screen.
When reading is complete, the file is closed and execution terminates.

LIST command

FUNCTION

Outputs a program list to the screen or specified output file.

FORMAT

1 LIST [<starting Line No.>] [$\left|\begin{array}{c}-\\,\end{array}\right|$ [<ending point Line No.>]]

2 LIST "<file descriptor>" [,[<starting Line No.>] [$\left|\begin{array}{c}-\\,\end{array}\right|$ <ending Line No.>]]]

(The abbreviated form is L.)

STATEMENT EXAMPLE

```
LIST 10–200
```

EXPLANATION

A part or all of a BASIC program in the memory at present is output to the screen in the case of Format 1 and to the specified output file in the case of Format 2 . When the two line numbers are not specified, the whole program is output. When the specification for <starting Line No.> is omitted, the contents from the start of the program to the specified ending Line No. are output. When the specification for <ending Line No.> is omitted, the contents from the specified starting point up to the end of the program are output.
Both lines numbers need not actually exist.
Also, a period 「.」 may be used instead of a Line No. for both, and if a period is specified, a Line No. pointer that has been memorized by BASIC resulting from the execution of a LIST statement immediate before or the generation of an error is used.
As soon as a LIST command is executed, the computer goes back to the command level.

NOTE

To specify LPT1, LPT2 or COM1 ~ 4 as the device name in the file descriptor, a pertinent I/O card (expanded printer card MP-1810 or expanded RS-232C card MP-1820) is needed for the expanded interface.
When a cassette is specified for the device name, the result is the same as executing a SAVE command in character form. At this time, specification of the file name cannot be omitted.

REFERENCE

LLIST

SAMPLE PROGRAM

```
Ready
LIST ····························(1)

100 'LINE-1
200 'LINE-2
300 'LINE-3
400 'LINE-4
500 'LINE-5
600 'LINE-6
700 'LINE-7
800 'LINE-8
900 'LINE-9

Ready

LIST 300-500 ········(2)

300 'LINE-3
400 'LINE-4
500 'LINE-5

Ready

LIST 700- ················(3)

700 'LINE-7
800 'LINE-8
900 'LINE-9

Ready

LIST -300 ················(4)

100 'LINE-1
200 'LINE-2
300 'LINE-3

Ready
```

This is an example of a LIST instruction.

(1) is the display when no specification is given. The whole program is output.

(2) is the display when "300" is specified as the <starting Line No.> and "500" is specified for the <ending Line No.>. A list of the program from Lines 300 to 500 are output.

(3) is an example of specifying "700" as the <starting Line No.>, a minus sign 「−」 is specified after that and the specification of <ending Line No.> is omitted. Lines 700 through 900 are output.

In (4), the specification of <starting Line No.> is omitted, a minus sign 「−」 is specified, and "300" is specified after it as the <ending Line No.>. The list shows the program from the start to Line 300.

LLIST command

FUNCTION Outputs a program list to a printer.

FORMAT LLIST [<starting Line No.>] [|- ,| [<ending Line No.>]]
(The abbreviated form is LL.)

STATEMENT EXAMPLE LLIST 10–200

EXPLANATION

A part or all of a BASIC program in the memory at present is output to a printer (LPT 0:).
When the specifications of the two line numbers are omitted, the whole program is output. When the specification for the <starting Line No.> is omitted, the program from the program start to the specified ending Line is output. When the specification of the <ending Line No.> is omitted, the program from the starting Line No. to the end of the program is output.
Both of these specified lines need not actually exist.
Also, a period 「.」 may be used instead of a Line No. for both, and if a period is used, a Line No. pointer that has been memorized by BASIC resulting from the execution of a LIST statement immediate before or the generation of an error is used.
As soon as an LLIST command is executed, the computer goes back to the command level.

REFERENCE

LIST

SAMPLE PROGRAM

Refer to the sample program of the LIST command.

LOAD command

FUNCTION	Reads a program stored on cassette tape.
FORMAT	LOAD ["[CAS0:] [<file name>]" [, R]]
	(The abbreviated form is LO.)
STATEMENT EXAMPLE	LOAD "PROG1"

EXPLANATION

This statements loads a program file having the name specified by <file name> to memory from a cassette tape. When the file name specification is omitted, the first file found on the tape is loaded. Programs in the memory are not erased until the specified file is found. Key in CTRL + D to stop the operation when the specified tape is being searched.
When option "R" is specified, the specified program file on the cassette tape is loaded to the memory and the program is immediately executed. Also, all files that are open at the time are kept open.

NOTE

When an input is made during LOADing from a cassette tape, the computer may not operate normally sometimes. Do not input anything during the LOAD operation.

REFERENCE

SAVE, MERGE, RUN

LOAD? command

FUNCTION	Checks a file on a cassette tape.
FORMAT	LOAD? ["[CAS0:] [<file name>] "] (The abbreviated form is LO. ?)
STATEMENT EXAMPLE	LOAD? "PROG1"

EXPLANATION

When a program or data is recorded on a cassette tape, checksum data, which is provided to check correct loading, is recorded on the cassette tape together with the program or data.
This statement calculates a checksum of a program file or data file on the tape, compares it with the recorded checksum, thereby checking whether or not the program file or data file is correctly recorded on the tape. If the two checksums match, "Ready" is displayed. If the two do not match, it results in a Device I/O Error.
When the <file name> is specified, the file is checked. If it is omitted, the first file found is checked.

NOTE

(1) This statement does not compare a file with a file in memory. Also, even if the file checked is a program file, the file is not loaded.
(2) If a LOAD? instruction is executed during a program, when LIST is output, LOAD-PRINT is displayed.

LOAD command

FUNCTION	Reads a machine language program recorded on a cassette tape.
FORMAT	LOADM ["[CAS0:] [<file name>]" [,[<offset>] [,R]]] (The abbreviated form is LO.M)
STATEMENT EXAMPLE	LOADM "PROG1",, R

EXPLANATION

This statement reads a machine language program, etc. (items recorded by a SAVEM command) recorded on a cassette tape into memory.
When the specification <offset> is given, the program is read into an address which is a sum of the address at the time of recording and the offset value. Specify a value in the range –65536 ~ 65535.
When option "R" is specified, the machine language program is executed immediately after it is read. The execution start address is the <starting address> specified by the SAVEM command.
When the <file name> is not specified, the first file found on the tape is loaded. Therefore, if the file is not a file recorded by a SAVEM command, it results in an error.

REFERENCE

SAVEM

LOCATE

FUNCTION Specifies the cursor position and function.

FORMAT LOCATE <X>, <Y> [,<cursor mode>]
(The abbreviated form is LOC.)

STATEMENT EXAMPLE LOCATE 20, 12

EXPLANATION

This statement moves the cursor to the specified position. The position is the position of screen in the character mode. Specify <X> and <Y> in the range shown in the table below. The reference point at the upper left corner.

Number of characters displayed horizontal / Parameter	40 characters	80 characters
<X>	0 ~ 39	0 ~ 79
<Y>	0 ~ 24	

The cursor state can be changed by specifying the <cursor mode>.

<cursor mode>	Details
0	Non-blinking (The cursor is displayed but it does not blink.)
1	No cursor (The cursor is not displayed.)
2	Double-speed blinking (The cursor is displayed and the blinking speed is twice as fast as when specifying 「3」.)
3	Normal blinking (The cursor is displayed and it blinks.)

When the <cursor mode> is not specified, the preceding mode is used.

REFECENCE

SCREEN function

SAMPLE PROGRAM

```
LIST

100 'LOCATE statement
110 CLS :WIDTH 80
120 FOR X=0 TO 60 STEP 20
130   FOR Y=0 TO 12 STEP 3
140     LOCATE X,Y
150     PRINT "LOCATE";X",";Y
160   NEXT Y
170 NEXT X
180 END

Ready
```

```
LOCATE 0 , 0        LOCATE 20 , 0       LOCATE 40 , 0       LOCATE 60 , 0

LOCATE 0 , 3        LOCATE 20 , 3       LOCATE 40 , 3       LOCATE 60 , 3

LOCATE 0 , 6        LOCATE 20 , 6       LOCATE 40 , 6       LOCATE 60 , 6

LOCATE 0 , 9        LOCATE 20 , 9       LOCATE 40 , 9       LOCATE 60 , 9

LOCATE 0 , 12       LOCATE 20 , 12      LOCATE 40 , 12      LOCATE 60 , 12
Ready
```

This program displays characters while the cursor position is changed by the LOCATE statement on Line 140.
The coordinates displayed on the screen are the coordinates of the position of "L" in "LOCATE".

LPRINT, LPRINT USING

FUNCTION	Outputs data to a printer.
FORMAT	1 LPRINT [<expression>] [;<expression>] ... [;] 2 LPRINT USING <format control character string> ; <expression> [;<expression>] ... [;]
STATEMENT EXAMPLE	1 LPRINT "TEST" 2 LPRINT USING "###.#" ; A

EXPLANATION

An LPRINT or LPRINT USING statement is almost the same as a PRINT or PRINT USING statement, except that an LPRINT or LPRINT USING statement outputs data to a printer.
The maximum output width of a printer is 80 characters a line, and regardless of the number of characters that can be printed by the printer connected, when a line is printed with this maximum printing width, line is feed occurs.
This value can be changed by the specification of WIDTH "LPT0:", <number of printable characters>.

NOTE

An LPRINT statement can output "\" (back slash) which cannot be output by a PRINT statement.

REFERENCE

WIDTH, TAB function, SPC function, PRINT, PRINT USING

SAMPLE PROGRAM

```
Ready
LIST

100 'LPRINT statement
110 OPEN"O",#1,"TEST"
120 A$="ABC":B$="DEF"
130 PRINT #1,A$;B$
140 CLOSE:STOP
150 OPEN "I",#1,"TEST"
160 INPUT #1,Y$,Z$
170 LPRINT Y$
180 LPRINT Z$
190 CLOSE
200 END

Ready

RUN

Break In 140
Ready
CONT

Ready

ABC
DEF
```

Lines 170 and 180 output character strings "ABC" and "DEF" to a printer.

LSET, RSET

FUNCTION	Left or right justifies character type variables.
FORMAT	1 LSET <character type variable> = <character string> 2 RSET <character type variable> = <character string>
STATEMENT EXAMPLE	1 LSET X$= "SAMPLE" 2 RSET Y$= "TEST"

EXPLANATION

An LSET statement sets the specified <character string> to the <character type variable> with left justification.
An RSET statement sets the specified <character string> to the <character type variable> with right justification.
When using an LSET or RSET statement, an arbitrary character string must be assigned to the <character type variable> in advance.
If the <character type variable> is longer than the <character string>, blanks are left at the left or right, respectively. If the <character string> is longer than the <character type variable>, the right side of <character string> is lost.

	LSET	RSET
When character type variable (A$) is longer than character string, A$ = "XXXXXX" is set.	When LSET A$ = "ABC" is executed, A$ = "ABC␣␣␣" is set.	When RSET A$ = "ABC" is executed, A$ = "␣␣␣ ABC" is set.
When character type variable (A$) is shorter than character string, A$ = "XXX" is set.	When LSET A$ = "ABCDE" is executed, A$ = "ABC" is set.	When RSET A$ = "ABCDE" is executed, A$ = "ABC" is set.

REFERENCE

MKI$, MKS$, MDK$

SAMPLE PROGRAM

```
Ready
LIST

100 ' LSET,RSET statement
110 A$="        ":B$="
120 LSET A$="A"
130 RSET B$="B"
140 PRINT A$
150 PRINT B$
160 END

Ready
RUN
A
       B

Ready
```

This program outputs a list of left and right justified character strings.

MERGE command

FUNCTION

Merges a program recorded on a cassette tape with a program in memory.

FORMAT

MERGE ["[CAS0:] [<file name>]" [,R]]
(The abbreviated form is ME.)

STATEMENT EXAMPLE

MERGE "CAS0: PROG1", R

EXPLANATION

This statement merges a program with the specified file name (the first file if the file descriptor is omitted) on a cassette tape to a BASIC program in the memory at present. (The term of "merge" may be interpreted as "inserting" a program on a cassette tape into a program in memory.)
When a program contains a MERGE command, the computer waits for a command after executing the MERGE command. However, if option "R" is specified, the execution starts from the program start after executing the MERGE command.

NOTE

A program file that is used with a MERGE command must be a program that has been saved in character form.
If the program on the cassette tape has a line with the same Line No. as that of the program in memory, the contents on the program line in memory are replaced by the contents of the line of the program on the tape.

REFERENCE

SAVE, LOAD

SAMPLE PROGRAM

```
Ready
LIST

100 'MERGE statement  }
110 PRINT "DEF"       } ................(1)
120 END               }

Ready
SAVE"TEST",A

Ready
NEW

Ready
110 PRINT "ABC"       }
120 MERGE "TEST"      } ................(2)
130 END               }
RUN
ABC

Ready


RUN
DEF

Ready
LIST

100 'MERGE statement
110 PRINT "DEF"
120 END
130 END

Ready
```

First, Program (1) consisting of Lines 100 to 120 is saved with Option A. Then, a NEW command is executed and Program (2) consisting of Lines 110 to 130 is input. When Program (2) is run, the character string "ABC" is printed out. Since a MERGE command is executed by Line 120 of Program (2), Line 110 of (2) is replaced by Line 120 of (1). When the program is run after executing the MERGE command, this time "DEF" is printed out.
If LIST is instructed, a list is output which shows how the MERGE command was executed.

[illegible]

MID$

FUNCTION — Replaces part of a character variable.

FORMAT — MID$ (<character variable name>, <argument 1> [,<argument 2>]) = <character expression>

STATEMENT EXAMPLE — MID$ ("ABCD", 2, 2) = "Y2"

EXPLANATION

This statement replaces the characters starting at the character with the <argument 1> order in the specified character variable for the number of characters specified by <argument 2> by the contents of the <character expression>. If the length of <character expression> is longer than the <argument 2>, the excess on the right side is ignored. The number of characters to be replaced is the least among three values, <argument 2>, the length after the <argument 1> character of the <character variable name>, and the length of <character expression>. When the <argument 2> specification is omitted, the least of the other two is used.
The value of <argument 2> must be in the range 0 ~ 255. <argument 1> must be specified as a value in the range 1 ~ 255 and the length must not exceed the length of the specified character variable.

REFERENCE

MID$ function

SAMPLE PROGRAM

```
Ready
LIST

100 'MID$ statement
110 A$="ABCDEFG"
120 PRINT A$
130 B$="WXYZ"
140 MID$(A$,5,2)=B$
150 PRINT A$
160 END

Ready
RUN
ABCDEFG
ABCDWXG

Ready
```

Lines 110 and 130 assign character strings to A$ and B$ and Line 120 prints out A$. Line 140 replaces the two characters starting with the fifth character in A$ with the first two characters of B$. A$ is printed out after the MID$ operation.

MON command

FUNCTION Transfers execution to the machine language monitor.

FORMAT MON

EXPLANATION

When MON is input when waiting for an instruction, the computer is set to the monitor mode. In the monitor mode, the contents of memory and registers can be referred to or altered or execution can be started at any address by inputting a subcommand.
The monitor mode is terminated when a Q subcommand is executed, and the computer goes back to the "waiting for an instruction" state.
When the monitor mode is set, a prompt symbol 「*」 is displayed and the following subcommands can be input.

Subcommand name	Function
B, BL	Displays and sets break points.
BP	Displays break points.
D	Displays memory contents.
DL	Sets the number of lines displayed by Subcommand D.
E	Displays and alters the contents of memory by the screen editor.
F	Fills up memory with a certain data pattern.
G	Executes a machine language program.
H	Displays the sum and difference of hexadecimal numbers.
HC	Sets the hard copy mode.
M, ML	Displays and alters the contents of memory.
MAP, MAPL	Displays and alters the current memory area No. or the map image(Note) of the specified memory area.
MAPP	Displays the current memory area No. or the map image of the specified memory area.
Q	Terminates the monitor mode.
R, RL	Displays and alters the contents of a register.
RP	Displays the contents of a register.
S	Retrieves a certain data pattern from memory.
T	Transfers the memory contents to an other address.
TS	Sets the text screen save/recovery mode.
X, XL	Displays and alters the map image of the current memory area.
XP	Displays the map image of the current memory area.

(Note) Map image: Setting state of the address mapping register of the memory area. (For details, refer to the hardware manual.)

Input a subcommand following the prompt sign 「*」 and with the following format:

```
<subcommand name> ␣ <operand 1>, <operand 2>, ...
```

「␣」 represents a space.

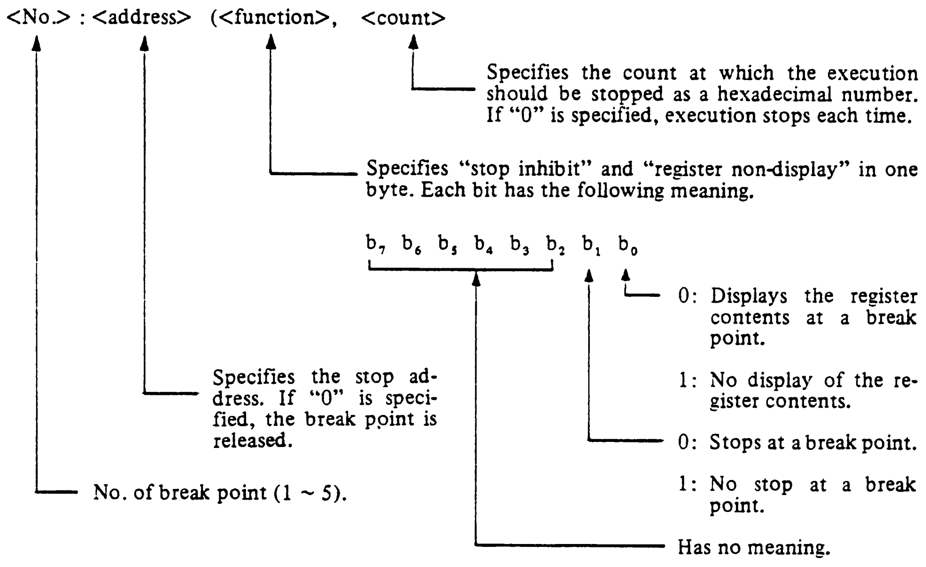
Specify the <operand> in the subcommand format with a hexadecimal number of up to four digits, 「$」 or 「*」. When 「$」 is specified, the program counter value at the start is taken. When 「*」 is specified, the program counter values that is currently set are taken.
Specify the ≪operand≫ as a hexadecimal number of any number of digits or a character string (enclosed by double quotation marks 「"」).
When the [BREAK], [CTRL] + [C], [CTRL] + [D], or [BREAK] + reset keys are pressed during execution of a subcommand, the subcommand execution is temporarily stopped and the prompt sign 「*」 is displayed.

(1) Subcommand B or BL

FUNCTION	Displays and sets a break point.
FORMAT	1 B 2 BL

EXPLANATION

Execution of a machine language program can be stopped at any address. This subcommand displays the place (break point) at which the execution is stopped and the point can be set again later. Up to five break points can be set and they are displayed on the screen in the following format.



```
<No.> : <address> (<function>, <count>)
```

Specifies the count at which the execution should be stopped as a hexadecimal number. If "0" is specified, execution stops each time.

Specifies "stop inhibit" and "register non-display" in one byte. Each bit has the following meaning.

$b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$

0: Displays the register contents at a break point.
1: No display of the register contents.
0: Stops at a break point.
1: No stop at a break point.
Has no meaning.

Specifies the stop address. If "0" is specified, the break point is released.

No. of break point (1 ~ 5).

When Subcommand B is used, the contents before and after alteration are displayed simultaneously so the two can be compared and checked.
When Subcommand BL is used, the altered contents are displayed on the same line as the contents before alteration.

NOTE

(1) A break point cannot be set in the ROM area.

(2) An break point instruction set by a B or BL subcommand is replaced by an SWI instruction (Operation code = 3F) by Subcommand G when a machine language program is executed, and when the execution stops, it returns to the initial instruction. Since the initial instruction is executed when Subcommand G is specified again, even if a break point is specified, the instruction is not replaced with an SWI instruction. (In other words, merely setting a break point at the starting address of subcommand G has no meaning.) Therefore, the <count> is specified so as the execution is stopped at an arbitrary count, or, to stop execution twice or more at the same break point, a pair of dummy break points must be specified.

USE EXAMPLE

```
*D 5000,1
5000  4A 5A 7E 50 00 FF 00 00 - FF FF 00 00 FF FF 00 00 (6D)   JZ~P. ..  ..  ..
*BP
1:5001(00,0010) 2:5002(03,0000) 3:0000(00,0000) 4:0000(00,0000) 5:0000(00,0000)
*G 5000
S=E45D(F200)CC=88(N***)A=8F B=38 DP=00 X=EAD9 Y=F3B5 U=C066 PC=5001(5A7E5000FF)
*G
S=E45D(F200)CC=88(N***)A=AF B=28 DP=00 X=EAD9 Y=F3B5 U=C066 PC=5001(5A7E5000FF)
*G
S=E45D(F200)CC=88(N***)A=9F B=18 DP=00 X=EAD9 Y=F3B5 U=C066 PC=5001(5A7E5000FF)
*G
S=E45D(F200)CC=88(N***)A=8F B=08 DP=00 X=EAD9 Y=F3B5 U=C066 PC=5001(5A7E5000FF)
```

1 The following program is stored starting at Address 5000:

<address>	<contents>	<assembler mnemonic>
5000	4A	DECA
5001	5A	DECB
5002	7E5000	JMP $5000

2 Break points are set as follows:
At Address 5001, execution stops once every 16th count and the register contents are displayed.
At Address 5002, the execution does not stop and the register contents are not displayed (dummy break point).

3 When a machine language program is executed from Address 5000, the execution stops when it reaches Address 5001 16 times, and the register contents are displayed.

4 After that, if a machine language program is executed by subcommand G, the execution stops at Address 5001 once every 16th count, and the register contents are displayed.

(2) **Subcommand BP**

FUNCTION Displays the contents of current break points.

FORMAT BP

EXPLANATION

This subcommand displays the contents of five break points that are set at present. All values (address, function and count) are set to "0" when the monitor mode starts.

(3) **Subcommand D**

FUNCTION Displays memory contents.

FORMAT D [<display address>], [<number of display lines>]

EXPLANATION

This subcommand displays the memory contents starting at the <display address> for the <number of display lines> as 16 bytes for one line. A value of "displayed address + 1" is memorized by the monitor.
The <display address> is shown as a four-digit hexadecimal number. When the display address is omitted, the contents are displayed from the address memorized by the monitor.
When the <number of display lines> is omitted, the contents are displayed for the number of display lines set by Subcommand DL. If Subcommand DL is not being executed, the contents are displayed for one line only.

USE EXAMPLE

```
*D 1FFF,2
1FFF  FF FF FF 00 00 FF FF 00 - 00 FF FF 00 00 FF FF 00 (F7)     ..  ..  ..  .
200F  00 FF FF 00 00 FF FF 00 - 00 FF FF 00 00 FF FF 00 (F8)  .  ..  ..  ..  .
*
```

(4) **Subcommand DL**

FUNCTION Sets the number of lines displayed by Subcommand D.

FORMAT DL <number of lines>

EXPLANATION

This subcommand displays the memory contents for the set number of lines. If this is not specified, the contents are displayed for the <number of lines> set by the main command.

USE EXAMPLE

```
*DL 5
*D 1FFF,
1FFF  FF FF FF 00 00 FF FF 00 - 00 FF FF 00 00 FF FF 00 (F7)     ..  ..  ..  .
200F  00 FF FF 00 00 FF FF 00 - 00 FF FF 00 00 FF FF 00 (F8)  .  ..  ..  ..  .
201F  00 FF FF 00 00 FF FF 00 - 00 FF FF 00 00 FF FF 00 (F8)  .  ..  ..  ..  .
202F  00 FF FF 00 00 FF FF 00 - 00 FF FF 00 00 FF FF 00 (F8)  .  ..  ..  ..  .
203F  00 FF FF 00 00 FF FF 00 - 00 FF FF 00 00 FF FF 00 (F8)  .  ..  ..  ..  .
*
```

(5) **Subcommand E**

FUNCTION	Displays and alters the contents of memory by the screen editor.
FORMAT	E [<display address>]

EXPLANATION

This subcommand displays the contents of memory for 256 bytes from the <display address>, and the cursor blinks at the first byte on the upper left. To change the memory contents, move the cursor by operating the cursor keys ([↑][↓][←][→]) and input a hexadecimal number or character. The following key operations are possible.

[CTRL] + [A]	....	This changes the editing mode (hexadecimal/character).
[CTRL] + [B] or [SHIFT] + [←]	....	Displays the memory contents of the 256 bytes before the current display.
[CTRL] + [F] or [SHIFT] + [→]	....	Displays the memory contents of the 256 bytes after the current display.
[CTRL] + [G]	....	Sets a break point to the cursor position. The contents of the break point are: Break point No. = 5, Function = Stop, Register display is specified, and Count = 0.
[↵]	....	Terminates Subcommand E.

NOTE

The screen display by Subcommand E is not output to a printer if printer output is specified by a Subcommand HC.

(6) **Subcommand F**

FUNCTION	Fills up a memory with a certain data pattern.
FORMAT	F <start address>, $\left\{ \begin{array}{l} \text{<end address>} \\ \text{@<number of bytes>} \end{array} \right\}$, ≪data pattern≫

EXPLANATION

This subcommand repeatedly fills the area from the specified <start address> to the specified <end address> or from the specified <start address> for the specified <number of bytes> with the specified ≪data pattern≫.

USE EXAMPLE

```
*F 1F00,@20,ABCD
*D 1F00,3
1F00  AB CD AB CD AB CD AB CD - AB CD AB CD AB CD AB CD (C0)  π^π^π^π^π^π^π^π^
1F10  AB CD AB CD AB CD AB CD - AB CD AB CD AB CD AB CD (C0)  π^π^π^π^π^π^π^π^
1F20  00 00 FF FF 00 00 FF FF - 00 00 FF FF 00 00 FF FF (F8)  ..  ..  ..  ..
*
```

(7) **Subcommand G**

FUNCTION

Executes a machine language program.

FORMAT

G [<execution start address>]

EXPLANATION

This subcommand transfers the control to the specified <execution start address>. When the specification of the <execution start address> is omitted, the execution starts at the address held by the program counter.
When $\boxed{\text{BREAK}}$, $\boxed{\text{CTRL}}+\boxed{\text{C}}$, $\boxed{\text{BREAK}}$ + reset button are pressed during execution of this command, the register of before executing this command is displayed and a prompt sign 「*」 is displayed. When $\boxed{\text{CTRL}}+\boxed{\text{D}}$ are pressed, the current register value is displayed.

NOTE

If there is no machine language program at the execution start address, or if the program is not described correctly, the execution of Subcommand G may cause abnormal operation of the computer or other programs may sometimes be erased.

(8) Subcommand H

FUNCTION

Displays the sum and difference of hexadecimal numbers.

FORMAT

H <value 1>, <value 2>

EXPLANATION

Specify <value 1> and <value 2> using hexadecimal numbers of up to four digits. The sum and difference of <value 1> and <value 2> are calculated and the results are displayed as two bytes. If the addition result is greater than two bytes, the lower two bytes are displayed. If the subtraction result is a negative value, the 2's complement is displayed.

USE EXAMPLE

$$\texttt{*H }\overbrace{\texttt{ABCD}}^{\text{Value 1}}\texttt{,}\overbrace{\texttt{1111}}^{\text{Value 2}}$$

$$\underbrace{\texttt{BCDE}}_{\text{Addition result (= Value 1 + Value 2)}}\ \underbrace{\texttt{9ABC}}_{\text{Subtraction result (= Value 1 - Value 2)}}$$

```
*
```

(9) Subcommand HC

FUNCTION

Sets the hard copy mode.

FORMAT

HC [<hard copy mode>]

EXPLANATION

This subcommand specifies whether or not the contents displayed on the screen are to be output to a printer. When "0" is specified for the <hard copy mode>, there is no output to a printer. When "1" is specified, the contents are output to a printer.
When the specification of the <hard copy mode> is omitted, the mode is inverted.

(10) Subcommand M or ML

FUNCTION	Displays and alters the memory contents.
FORMAT	1 M [<memory address>]
	2 ML [<memory address>]

EXPLANATION

This subcommand displays the memory contents for 16 bytes from the specified <memory address>, and when necessary, the memory contents can be altered. When the specification of <memory address> is omitted, the display starts at the address obtained by adding one to the last address displayed by the monitor. When Subcommand M is used, the contents before and after alteration are displayed simultaneously so the two can be compared. When Subcommand ML is used, the altered contents are displayed on the same line as the contents before alteration.

NOTE

If a program other than one written in user machine language is altered by Subcommand M or ML, the computer may sometimes become unable to operate normally.

USE EXAMPLE

```
*M 1F00
1F00  AB CD AB CD AB CD AB CD - AB CD AB CD AB CD AB CD
      12 34 56 78 90 AB CD EF   12 34 56 78 90 AB CD EF
1F00  12 34 56 78 90 AB CD EF - 12 34 56 78 90 AB CD EF
*
```

(11) Subcommand MAP or MAPL

FUNCTION	Displays and alters the current memory area No. or the map image of the specified memory area.
FORMAT	1 MAP [<memory area No.>] 2 MAPL [<memory area No.>]

EXPLANATION

This subcommand displays and alters the map image contents of the memory area specified by the <memory area No.> (pages D through F cannot be altered). When the specification of <memory area No.> is omitted, the current memory area No. is displayed and altered. When the current memory area No. is altered, subcommands such as D or E reference/alte the memory in the memory area after alteration. Memory area numbers from "0" to "6" are reserved by the system, and if their map images are changed, the system may not normally operate.
The map image of user memory area is in memory area F when DEFMAP = 0 and in memory area E when DEFMAP = 1 ~ 3. When Subcommand MAP is used, the contents before and after alteration are displayed simultaneously and the two can be compared. When Subcommand MAPL is used, the altered contents are displayed on the same line as the contents before alteration.

USE EXAMPLE

(1) The following alters the current memory area.

```
*MAP
CURRENT MAP = 0F
              01
CURRENT MAP = 01
*
```

(2) Alteration of map image.

```
*MAP OF
0=03 1=04 2=05 3=06 4=07 5=08 6=09 7=0A 8=0B 9=0C A=0D B=0E C=0F D=85 E=84 F=EF
  BF   BF
0=BF 1=BF 2=05 3=06 4=07 5=08 6=09 7=0A 8=0B 9=0C A=0D B=0E C=0F D=85 E=84 F=EF
*
```

(12) Subcommand MAPP

FUNCTION	Displays the current memory area No. or the map image of the specified memory area.
FORMAT	MAPP [<memory area No.>]

EXPLANATION

This subcommand displays the map image contents of the memory area specified by the <memory area No.>.
When the specification of <memory area No.> is omitted, the current memory area No. is displayed.

(13) Subcommand Q

FUNCTION	Terminates the monitor mode.
FORMAT	Q

(14) Subcommand R or RL

FUNCTION	Displays and alters register contents.
FORMAT	1 R 2 RL

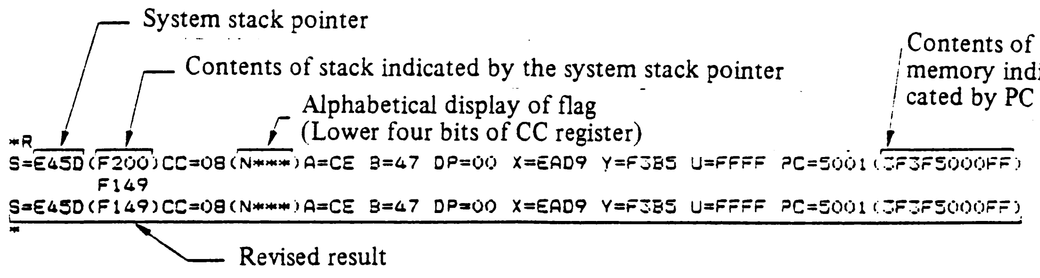
EXPLANATION

This subcommand displays the MPU register contents and changes them when necessary.
The display contents are as follows:

S :	Two bytes of system pointer and stack contents
CC:	Condition code (alphabetical display of NZVC flag when it is on)
A :	Accumulator A
B :	Accumulator B
DP:	Direct page register
X :	Index register X
Y :	Index register Y
U :	User stack pointer
PC:	Five bytes of program counter and memory contents

When subcommand R is used, the contents before and after alteration are displayed simultaneously so the two can be compared. When subcommand RL is used, the altered contents are displayed on the same line as the contents before alteration.

USE EXAMPLE


System stack pointer
Contents of stack indicated by the system stack pointer
Alphabetical display of flag (Lower four bits of CC register)
Contents of memory indicated by PC
Revised result

```
*R
S=E45D(F200)CC=08(N***)A=CE B=47 DP=00 X=EAD9 Y=F3B5 U=FFFF PC=5001(3F3F5000FF)
      F149
S=E45D(F149)CC=08(N***)A=CE B=47 DP=00 X=EAD9 Y=F3B5 U=FFFF PC=5001(3F3F5000FF)
*
```

(15) RP

FUNCTION Displays register contents.

FORMAT RP

EXPLANATION

This subcommand displays the same contents as Subcommands R and RL.

(16) Subcommand S

FUNCTION Retrieves a certain data pattern from memory.

FORMAT S [<start address>], [{ <end address>
@<number of bytes> }],
≪data pattern≫

EXPLANATION

This subcommand retrieves the contents as the specified ≪data pattern≫ from the specified memory range and displays the start address.
There are two ways to specify the range; one is from the <start address> to the <end address> and the other is the <number of bytes> from the <start address>.
When the specification of the <start address> is omitted, "0" is assumed. When the specification of the <end address> or <number of bytes> is omitted, hexadecimal number FDFF is assumed as the <end address>.
Specify the data to be retrieved as the ≪data pattern≫.

USE EXAMPLE

```
*D 1F00
1F00  FF B7 E6 71 B6 EC 59 27 - E0 8D BD 39 B6 E6 4F 26 (A3)    ｷ｢qｶ●Y'=■ｽ9ｶ｢O&
1F10, 05 7A E6 50 27 01 39 8E - E6 50 86 3C A7 84 EC 05 (B8)   .z｢P'.9■｢P■<ｧ■●.
1F20  5C E7 06 C1 0A 25 07 6F - 06 4C A7 05 81 06 10 25 (69)   ¥ｧ.ｷ.%.o.Lｧ.＿..%
1F30  00 7F 6F 05 EC 03 5C E7 - 04 C1 0A 25 74 6F 04 4C (4C)   ..o.●.¥ｧ.ｷ.%to.L
1F40  A7 03 81 06 25 6B 6F 03 - EC 01 5C E7 02 C1 0A 25 (55)   ｧ.＿.%ko.●.¥ｧ.ｷ.%
*S 1F00,1F40,064CA7
Found-----1F28
*
```

(17) Subcommand T

FUNCTION

Transfers the memory contents to another address.

FORMAT

T <start address>, $\left\{ \begin{array}{l} \text{<end address>} \\ \text{@<number of bytes>} \end{array} \right\}$, <destination address>

EXPLANATION

This subcommand copies the specified range of the memory contents to the specified <destination address>. There are two ways to specify the range, one is from the <start address> to the <end address> and the other is the <number of bytes> from the <start address>.

USE EXAMPLE

```
MON
*D CF00,5
CF00  FF B7 E6 71 B6 EC 59 27 - E0 8D BD 39 B6 E6 4F 26 (A3)    ｷ◥ｸｶ●Y'=■ｽ9ｶ◥□&
CF10  05 7A E6 50 27 01 39 8E - E6 50 86 3C A7 84 EC 05 (B8)   .z◥P'.9■◥P■(ｧ▬●.
CF20  5C E7 06 C1 0A 25 07 6F - 06 4C A7 05 81 06 10 25 (69)   ¥◤.ﾁ.%.o.Lｧ.▁.%
CF30  00 7F 6F 05 EC 03 5C.E7 - 04 C1 0A 25 74 6F 04 4C (4C)   ..o.●.¥◤.ﾁ.%to.L
CF40  A7 03 81 06 25 6B 6F 03 - EC 01 5C E7 02 C1 0A 25 (55)   ｧ.▁%ko.●.¥◤.ﾁ.%
*T CF00,CF07,CF30
*D CF00,5
CF00  FF B7 E6 71 B6 EC 59 27 - E0 8D BD 39 B6 E6 4F 26 (A3)    ｷ◥ｸｶ●Y'=■ｽ9ｶ◥□&
CF10  05 7A E6 50 27 01 39 8E - E6 50 86 3C A7 84 EC 05 (B8)   .z◥P'.9■◥P■(ｧ▬●.
CF20  5C E7 06 C1 0A 25 07 6F - 06 4C A7 05 81 06 10 25 (69)   ¥◤.ﾁ.%.o.Lｧ.▁.%
CF30  FF B7 E6 71 B6 EC 59 27 - 04 C1 0A 25 74 6F 04 4C (56)    ｷ◥ｸｶ●Y'.ﾁ.%to.L
CF40  A7 03 81 06 25 6B 6F 03 - EC 01 5C E7 02 C1 0A 25 (55)   ｧ.▁%ko.●.¥◤.ﾁ.%
*
```

(18) Subcommand TS

FUNCTION

Sets the save/recovery mode of the text screen.

FORMAT

TS [<test screen save flag>]

EXPLANATION

When other than "0" is specified as the <text screen save flag>, the text screen is saved at the time program execution is stopped at a break point, and the saved text screen is recovered at the time of executing Subcommand G.
When "0" is specified for the <text screen save flag>, the save/recovery operation of the text screen is not conducted.
When the specification of <text screen save flag> is omitted, the mode is inverted.

(19) Subcommand X or XL

FUNCTION	Displays and alters the map image of current memory area.
FORMAT	1 X 2 XL

EXPLANATION

This subcommand displays the map image of the current memory area and can change it if necessary. The result of executing this subcommand is the same as specifying the current memory area No. by Subcommand MAP or MAPL.

(20) Subcommand XP

FUNCTION	Displays the map image of current memory area.
FORMAT	XP

EXPLANATION

The result of executing Subcommand XP is the same as specifying the current memory area No. by Subcommand MAPP.

NOTE

Pay particular attention to changing any program other than a user machine language program when using a MON command. The computer may run out of control when a memory of other than a user machine language program is changed or the map image is changed.

MOTOR

FUNCTION

Controls the motor of the cassette tape recorder.

FORMAT

MOTOR [[<switch>] [,<baud rate>]]
(The abbreviated form is M.)

STATEMENT EXAMPLE

MOTOR ON, 1

EXPLANATION

This statement controls the motor of the cassette tape recorder.
When 「ON」 is specified for the <switch>, the motor is turned on and when 「OFF」 is specified, the motor is turned off. If the specification of <switch> is omitted, the current state is inverted (On → Off, Off → On).
If an error occurs in the ON state, the state is automatically set to OFF.
The <baud rate> is the cassette tape recorder data transfer speed. If this specification is omitted, the set value remains unchanged.

<baud rate>	Contents
0 (Initial value)	600 baud
1	1200 baud
2	2400 baud

NOTE

(1) Use cassette tape recorder TRQ-2400 at the 2400 baud rate.
(2) When the tape is recorded at the 2400 baud rate, use the same MB-S1 and cassette recorder for input.

MTRIG(n) ON/OFF/STOP

FUNCTION	Permits, inhibits or stops an interrupt from the mouse.
FORMAT	MTRIG (<button No.>) { ON / OFF / STOP }
STATEMENT EXAMPLE	MTRIG (1) ON

EXPLANATION

Specify the mouse button by the <button No.>. "1" is for button A and "2" for button B.

(1) A MTRIG(n) ON statement permits an interrupt from the mouse. After that, each time the specified mouse button is pressed, an interrupt occurs, and the execution branches to the interrupt processing routine defined by an ON MTRIG(n) GOSUB statement.

(2) A MTRIG(n) OFF statement inhibits interrupts from the mouse. After this, the execution does not branch to an interrupt processing routine even if a mouse button is pressed.

(3) A MTRIG(n) STOP statement stops interrupts from the mouse. When a mouse button is pressed, the computer registers that it has been pressed but no branching to an interrupt processing routine takes place.
However, if a MTRIG(n) ON statement is executed later, branching to the interrupt processing routine following the previous pressing of the button takes place.
A MTRIG(n) STOP statement is effective in the state of MTRIG(n) ON.

NOTE

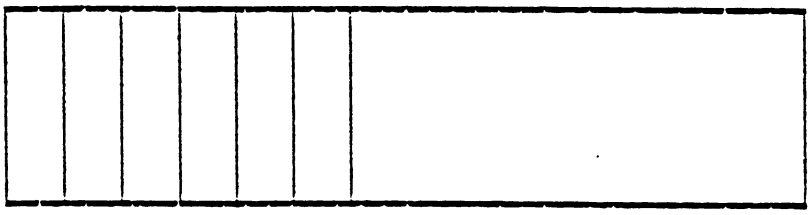
The MTRIG(n) OFF condition can be obtained by executing a command such as RUN, NEW and CLEAR.

REFERENCE

MREAD function, MTRIG function, ON MTRIG(n) GOSUB

SAMPLE PROGRAM

```
100 'MTRIG(n) ON/OFF/STOP statement
110 ON MTRIG(1) GOSUB 500
120 MTRIG(1) ON
130 PRINT" ﾏｳｽ ﾎﾞﾀﾝ A ｦ ｵｼﾃｸﾀﾞｻｲ !!! "
140 FOR I=1 TO 10000
150 NEXT I
160 MTRIG(1) OFF
170 PRINT" ﾏｳｽ ﾜﾘｺﾐｷﾝｼ !! "
180 FOR A=1 TO 5000
190 NEXT A
200 END
500 '
510 CLS
512 B=0
520 FOR S=1 TO 7
522 B=B+20
524 FOR S1=1 TO 100
530 LINE (B,70)-(300,100),PSET,S,B
532 NEXT S1
540 NEXT S
550 RETURN

Ready
RUN
 ﾏｳｽ ﾎﾞﾀﾝ A ｦ ｵｼﾃｸﾀﾞｻｲ !!!
```



```
 ﾏｳｽ ﾜﾘｺﾐｷﾝｼ !!

Ready
```

Line 120 permits an interrupt from the mouse. When mouse button A is pressed during execution of Lines 140 and 150, a rectangle is drawn in a different color due to the interrupt processing routine of Line 500. After this interrupts from the mouse are inhibited by Line 160.

NEW command

FUNCTION Erases a program and clears variables.

FORMAT NEW

EXPLANATION

This command erases BASIC in memory, clears all numeric variables to "0" and all character variables to null strings「" "」.
When this command is used in a program, the command is executed, program execution is stopped, and the computer is set to the state of waiting for a command.
The file opened remains in the open state.

SAMPLE PROGRAM

```
LIST

100 'NEW statement
110 A=10
120 PRINT A
130 END

Ready
RUN
 10

Ready
NEW

Ready
LIST


Ready
RUN

Ready
```

When a NEW command is executed after executing a program, nothing is displayed even if a LIST statement is executed. In other words, the program is erased.

NEW ON command

FUNCTION — Switches the system mode and restarts.

FORMAT — NEW ON <mode No.>

STATEMENT EXAMPLE — NEW ON 2

EXPLANATION

The specification of <mode No.> selects the graphic screen as well as the way to handle Hiragana characters at the time of printer output.
The <mode No.> has the following meaning:

O: Object of this command; ×: No object

Item / <mode No.>	Uses graphic screen	Changes Hiragana to Katakana outputs
0	○	×
1	×	×
2	○	○
3	×	○

If non-use of the graphic screen is specified by NEW ON 1 or NEW ON 3, the memory (48KB) for the graphic display can be used for text or variables. In such a case, graphic instructions (CIRCLE, LINE, etc.) cannot be used. Also, CLS2 only can be used on CLS, SCREEN,,0 or SCREEN,,1 only can be used on SCREEN.
When the power switch is turned on, the computer is set to the NEW ON 0 state.

ON COM(n) GOSUB

FUNCTION	Specifies the starting line of interrupt processing routine from a communications port.
FORMAT	ON COM (<port No.>) GOSUB <line No.>
STATEMENT EXAMPLE	ON COM (0) GOSUB 100

EXPLANATION

This statement specifies the Line No. to which program control is transferred when an input interrupt occurs at a communication port.
When an external input is given to the communication port specified by the <port No.>, execution is transferred to the processing subroutine specified by the <line No.>. After executing the processing subroutine, program control is returned by a RETURN statement to the place where the interrupt was applied in the main program. If a Line No. is specified by a RETURN statement, control is transferred to that line.

NOTE

To effect branching by an ON COM(n) GOSUB statement, the system must be set the ON state by a COM(n) ON/OFF/STOP statement in advance.

REFERENCE

COM (n) ON/OFF/STOP

SAMPLE PROGRAM

```
Ready
LIST

100 'ON COM(n) GOSUB statement
110 OPEN"I",#1,"COM0:"
120 ON COM(0) GOSUB 160
130 COM(0) ON : I=0
140 PRINT "WAITING FOR COM INPUT"
150 GOTO 150
160 ' INTRRUPT ROUTINE
170 I=I+1
180 INPUT #1,A$
190 IF I=100 THEN PRINT "END" :CLOSE #1: END
200 PRINT A$;
210 RETURN

Ready
*************************************************************************END

Ready
```

This program is an example of receiving an input from the RS-232C line.
Line 130 permits an interrupt. When an interrupt occurs in a loop because of Line 150, control is transferred to Line 160, and Line 180 receives the input.

ON ERROR GOTO

FUNCTION	Specifies the starting line of the error processing routine when an error occurs.
FORMAT	1 ON ERROR GOTO <line No.>
	2 ON ERROR GOTO 0
STATEMENT EXAMPLE	ON ERROR GOTO 100

EXPLANATION

When Format 1 has been instructed, if an error occurs during execution of a program, the function "displaying an error message on the screen and temporarily stopping execution" is cancelled, and execution is transferred to the error recovery processing routine starting at the specified <line No.>. By this function, temporary stopping of program execution caused by the generation of an error can be avoided.
This function can be released by executing Format 2 , a RUN command or a CLEAR command.
When executed in a main program, Format 2 releases the Format 1 instruction. When executed during an error recovery processing routine, it restores the function to temporarily stop the error recovery processing, displaying an error message on the screen and temporarily stops the program execution. Actual examples of using this function are shown below.
(1) An error for which no error recovery processing is needed.
(2) An error for which an error recovery processing should not be conducted.
(3) An error for which the contents of error recovery processing are not determined.

NOTE

This instruction is not effective for an error that occurs when waiting for a command or if it occurs during execution of an error recovery processing routine.
Also, on an error that occurs during execution of an INPUT or INPUT WAIT statement depending on the key-in contents, the computer outputs 「? Redo From Start」 and it waits for input. Therefore, no error processing is conducted.
If "0" is specified for the <line No.> of Format 1 , it acts as Format 2 . Line No. 0 cannot be specified as the starting line of an error processing routine.

REFERENCE

ERROR, ERL function, ERR function and RESUME

SAMPLE PROGRAM

```
Ready
LIST

100 'ON ERROR GOTO statement
110 ON ERROR GOTO 200
120 PRINT 1/0
130 END
200 PRINT "ERROR ! ERR NUMBER=";ERR
210 RESUME 130

Ready
RUN
ERROR ! ERR NUMBER= 11

Ready
```

Line 110 defines the error processing routine. When this program is run, execution jumps to Line 200 and an error message is output.

ON ~ GOTO/GOSUB

FUNCTION	Branches to a line No. or subroutine specified by the value of expression.
FORMAT	1 ON <expression> GOTO <line No.> [, <line No.>] ...
	2 ON <expression> GOSUB <line No.> [, <line No.>] ...
STATEMENT EXAMPLE	ON A + B GOTO 40, 50, 60

EXPLANATION

When Format 1 is instructed, execution branches to the line of <line No.> specified by the <expression>.
When Format 2 is instructed, execution branches to the subroutine <line No.> specified by the <expression>. Use a RETURN statement to return. For explanation of RETURN statement, refer to the GOSUB ~ RETURN statements.
The value of <expression> shows the order of line of <line No.> that is written after the GOTO or GOSUB statement. If <expression> is "2", the second <line No.> is the place to which branching is performed.
If the value of <expression> is "0" or larger than the number of lines specified, control is transferred to the next execution statement.

NOTE

(1) A negative value or a value greater than 256 cannot be given to the <expression>.
(2) For both formats 1 and 2 , a specification such as ⌜ON I GOSUB 10,,20⌟ omitting the intermediate line No. cannot be given.

REFERENCE

GOTO, GOSUB

SAMPLE PROGRAM

```
LIST

100 'ON n GOSUB statement
110 INPUT"INPUT 1-3 NUMBER:",A
120 ON A GOSUB 200,210,220
130 END
200 PRINT"ONE":RETURN
210 PRINT"TWO":RETURN
220 PRINT"THREE":RETURN

Ready
RUN
INPUT 1-3 NUMBER:2
TWO

Ready
```

This program prints out "ONE", "TWO" and "THREE" according to the input numerals of "1", "2" and "3". Line 110 inputs numeral to "A" and Line 120 causes a jump to a subroutine corresponding to the input value.
Each subroutine of Lines 200, 210 and 220 displays the character string on the screen, corresponding to each numeric input.

ON INTERVAL GOSUB

FUNCTION	Defines an interval timer interrupt processing routine.
FORMAT	ON INTERVAL GOSUB <line No.>
STATEMENT EXAMPLE	ON INTERVAL GOSUB 100

EXPLANATION

This statement defines a processing routine to which execution branches when an interval timer interrupt occurs.
Specify the number of the starting line for the interval timer interrupt processing routine for the <line No.>. Use a RETURN statement to return from the interval timer interrupt processing routine.

NOTE

In order for execution to branch by an ON INTERVAL GOSUB statement, the computer must be set to the ON state by an INTERVAL ON/OFF/STOP statement in advance.

REFERENCE

INTERVAL, INTERVAL ON/OFF/STOP

ON KEY(n) GOSUB

FUNCTION	Defines a starting line of processing subroutine when a programmable function key interrupt occurs.
FORMAT	ONKEY (<PF key No.>) GOSUB <line No.>
STATEMENT EXAMPLE	ON KEY (1) GOSUB 100

EXPLANATION

The specification of <PF key No.> is the same as that of a KEY instruction.
Define the line number to start the processing routine branched to when the PF key specified by the <PF key No.>. Supposing that "n" is the <PF key No.>, when the PFn key is pressed, control is transferred to the processing routine that starts at the <line No.> specified by the ON KEY(n) GOSUB statement.
Use a RETURN statement to return from the processing routine. When RETURN <line No.> is specified, execution restarts from this line. If it is not specified, the execution restarts at the place where execution was stopped.

NOTE

To branch with an ON KEY(n) GOSUB statement, the computer must be in the ON statement by a KEY(n) ON/OFF/STOP statement in advance.
This instruction can also be used as a command. However, since it is released when a RUN command is executed, execute a program by a GOTO statement for debugging a program. No interrupt is generated when waiting for a key input, but an interrupt does occur when an input occurs.

REFERENCE

KEY, KEY(n) ON/OFF/STOP

SAMPLE PROGRAM

```
Ready
LIST

100 'ON KEY (N) GOSUB statement
110 ON KEY(1) GOSUB 200
120 ON KEY(2) GOSUB 210
130 ON KEY(3) GOSUB 220
140 FOR I=1 TO 3:KEY (I) ON : NEXT I
150 PRINT "PRESS F1 TO F3"
160 GOTO 160
200 PRINT "F1 PRESSED!":RETURN 150
210 PRINT "F2 PRESSED!":RETURN 150
220 PRINT "F3 PRESSED!":RETURN 150

Ready
```

```
RUN
PRESS F1 TO F3
F1 PRESSED!
PRESS F1 TO F3
F2 PRESSED!
PRESS F1 TO F3
F3 PRESSED!
PRESS F1 TO F3

Break In 160
Ready
```

Lines 110 through 130 define the jump destination of each key and Line 140 permits key interrupts for F1 through F3. Execution enters an endless loop by Line 160, and the computer waits for a key input.
Lines 200 through 220 are the processing routine of each key, and here, the key that is pressed is displayed.

ON MTRIG(n) GOSUB

FUNCTION	Defines the starting line of a mouse interrupt processing routine.
FORMAT	ON MTRIG (<button No.>) GOSUB <line No.>
STATEMENT.EXAMPLE	ON MTRIG (1) GOSUB 100

EXPLANATION

This statement defines the processing routine that branches when an interrupt is generated as the specified mouse button is pressed.
Specify the mouse button by the <button No.>. "1" is for button A and "2" is for button B.
The <line No.> shows the starting line No. of the interrupt processing routine. If the computer is in the interrupt enabled state by a MTRIG(n) ON statement, when the mouse button is pressed, execution branches to the specified interrupt processing routine.
Use a RETURN statement to return from an interrupt processing routine.

NOTE

To effect branching by an ON MTRIG(n) GOSUB statement, the system must be set to the ON state by a MTRIG ON/OFF STOP statement in advance.

REFERENCE

MTRIG function, MREAD function, MTRIG(n) ON/OFF/STOP

SAMPLE PROGRAM

Refer to the sample program of MTRIG ON/OFF/STOP statement.

ON STRIG(n) GOSUB

FUNCTION	Defines the starting line of a joystick interrupt processing routine.
FORMAT	ON STRIG (<joystick No.>) GOSUB <line No.>
STATEMENT EXAMPLE	ON STRIG (1) GOSUB 100

EXPLANATION

This statement defines a processing routine to which the program branches when an interrupt occurs from the trigger button of the specified joystick.
Specify a joystick button by the <joystick No.>. "1" is for button A and "2" is for button B.
The <line No.> shows the starting line No. of the interrupt processing routine.
If a STRIG(n) ON statement has been executed and the system is in the interrupt permit state, when an interrupt from a joystick occurs, execution branches to the specified interrupt processing routine.
Use a RETURN statement to return from the interrupt processing routine.

NOTE

In order to effect branching by an ON STRIG(n) GOSUB statement, the system must be in the ON state with a STRIG(n) ON/OFF/STOP statement in advance.

REFERENCE

STICK function, STRIG function, STRIG(n) ON/OFF/STOP

SAMPLE PROGRAM

Refer to the sample program of the STRIG ON/OFF/STOP statement.

OPEN

FUNCTION	Allocates a buffer to an Input/Output file.
FORMAT	OPEN "<mode>", [#] <file No.>, "<file descriptor>"
STATEMENT EXAMPLE	OPEN "I", #1, "CAS0: TEST"

EXPLANATION

This statement allocates a file specified by the <file descriptor> to a buffer with the specified <file No.> and prepares for input/output, that is, it opens a file.
Specify either "I" or "O" for the <mode>. "I" is for the input mode and "O" is for the output mode. Specify an integer in a range of 1 ~ 16 for the <file No.>.
For details of the <file descriptor>, refer to the explanations (1) through (5) given below.
The ⌜"<mode>"⌟ and ⌜"<file descriptor>"⌟ can be specified as not only a Hollerith constant but also a character expression. The <file No.> can be specified with a numeric expression. An example is shown below. The example is only a part of program and applies to the part to open a file, and it must be followed by a suitable program.

Example:
```
10 M$= "O" : A=1
20 F$= "TEST"
30 OPEN M$, A, "CAS0 :"+F$
```

The format of "<file descriptor>" for each I/O device is explained below. The contents of Items 1 ~ 3 described after the device name are explained in the columns 1 ~ 3 in the same sequence.

1 Format of "<file descriptor>"
2 <mode> that can be specified
3 Other

(1) Keyboard
1 "KYBD:"
2 "I"
3 Two or more files can be opened by changing the specification of <file No.>.

(2) Screen
1 "SCRN:"
2 "O"
3 Two or more files can be opened by changing the specification of <file No.>.

(3) Printer

1 "LPT0:"

"LPT1:" }
"LPT2:" } Expansion printer card MP-1810 is necessary.

2 "O"

3 Two or more files can be opened by changing the specification of <file No.>.

(4) RS-232 port

1 "COM0: [(<option>)]"

"COM1: [(<option>)]" }
"COM2: [(<option>)]" } Expansion RS-232C card MP-1820 is necessary.
"COM2: [(<option>)]" }
"COM3: [(<option>)]" }

For the <option>, specify the conditions shown in the table below with up to four characters and enclosed by parentheses 「( )」. The specification for the whole of <option> can be omitted.

The characters usable for the <option> and the contents are shown in the table below.

Item No.	Item	Spec. char. for option	Contents	Default
1	Clock	F	fast (1/16 of clock)	S
		S	slow (1/64 of clock)	
2	Word length	7	7 bits/char.	8(Note)
		8	8 bits/char.	
3	Parity	E	Even parity	N
		O	Odd parity	
		N	Without parity	
4	Stop bit length	1	1 bit	2
		2	2 bit	

2 "I" or "O"

3 Two files can be opened for one port by changing the specifications for the <mode> and <file No.>.

4 Setting after the TERM mode is not affected.

(5) Cassette

1 "[CAS0:] <file name>"
  Specify the <file name> with characters and special signs up to eight characters. Some Hiragana characters and special signs cannot be used.

2 "I" or "O"

3 Only one file can be opened for a cassette.

REFERENCE

CLOSE

SAMPLE PROGRAM

```
LIST

100 'OPEN statement
110 OPEN "O",#1,"TEST"
120 PRINT #1,"ABCDEFG"
130 CLOSE
132 PRINT"ﾃｰﾌﾟ ｦ ﾏｷﾓﾄﾞｼﾃｸﾀﾞｻｲ !!"
140 STOP:FILES
150 END

Ready
RUN
ﾃｰﾌﾟ ｦ ﾏｷﾓﾄﾞｼﾃｸﾀﾞｻｲ !!

Break In 140
Ready
CONT

TEST        1 A
Abort In 140
Ready
```

This is an example of preparing a file using an OPEN statement. Line 110 opens a sequential file called "TEST", Line 120 writes a character string "ABCDEFG" to the file, and Line 130 closes the file. The written-to-file is displayed by the FILES statement on Line 140, and a file called TEST is prepared. The cassette tape must have been rewound when executing Line 140.

PAINT

FUNCTION

1 Paints the area enclosed by a specified boundary color using a specified color.

2 Fills the area enclosed by a specified boundary color using a specified tile pattern.

FORMAT

1 PAINT (<Wx, Wy>), [<area color>] [, <boundary color>]

2 PAINT (<Wx, Wy>), <tile string> [, <boundary color>]

STATEMENT EXAMPLE

1 PAINT (50, 50), 5, 4

2 PAINT (100, 50), CHR$ (&HAA) + CHR$ (&H55) + CHR$ (0), 3

EXPLANATION

1 This statement paints the area with the point specified by <Wx, Wy> with the specified <area color> using the color specified by the <boundary color> as the boundary.
Use the Palette No. system to specify the <area color> and <boundary color>. When the specification of <area color> is omitted, the Palette No. of the color specified by a COLOR statement is used as the <area color>. When the specification of <boundary color> is omitted, the Palette No. which is the same as that of the <area color> specification is used as the <boundary color>.
Painting starts at the point specified by <Wx, Wy>. This point must not be on the boundary. If it is on the boundary, painting is not performed.

2 This statement fills the area having the point specified by <Wx, Wy> with the tile pattern specified by the <tile string> using the color specified by the <boundary color> as the boundary.
The <tile string> is a character string that determines the tile pattern and tile size. The tile size in the horizontal direction is fixed to eight dots but the vertical direction size can be varied up to 36 dots depending on the length of the tile string. The tile pattern is determined by the bit pattern that corresponds to the <tile string> and the specification mechanism is that three characters constitute eight dots in the lateral direction and one dot in the vertical direction. The characters used in the string determine the dot patterns of blue, red and green planes in the order of the first to third characters from the head of the string. The character code determines which dot among the eight dots should be set or reset, or in other words, dots that correspond with the bit of "1" in the binary expression are set and dots that correspond with the bit of "0" in the binary expression are reset.

(Example) $\underbrace{\text{CHR\$ (\&HAA)}}_{\text{Definition of blue plane}} + \underbrace{\text{CHR\$ (\&H55)}}_{\text{Definition of red plane}} + \underbrace{\text{CHR\$ (0)}}_{\text{Definition of green plane}}$

The above example is a tile string that specifies a pattern of 8 x 1 dots. Since the data for the green plane is "0", green is not used and the tile pattern has alternative blue and red vertical lines. However, since the dots are very fine, no stripe effect is created, and the area looks purple.

Plane	Hexadecimal notation	Dot pattern
Blue	&HAA	⊘○⊘○⊘○⊘○
Red	&H55	○⊗○⊗○⊗○⊗
Green	&H00	○○○○○○○○
Screen display		⊘⊗⊘⊗⊘⊗⊘⊗

Since vertical lines are drawn in blue and red alternately, the overall appearance is purple.

By using the tiling, the basic eight colors can be mixed, creating intermediate colors.
If there is an excess in the tiling length, this part is ignored. A tiling statement consisting of only one or two characters is regarded an error.
When the <boundary color> is not specified, the Palette No. of the color specified by the COLOR statement immediately before is used as the <boundary color>.

NOTE

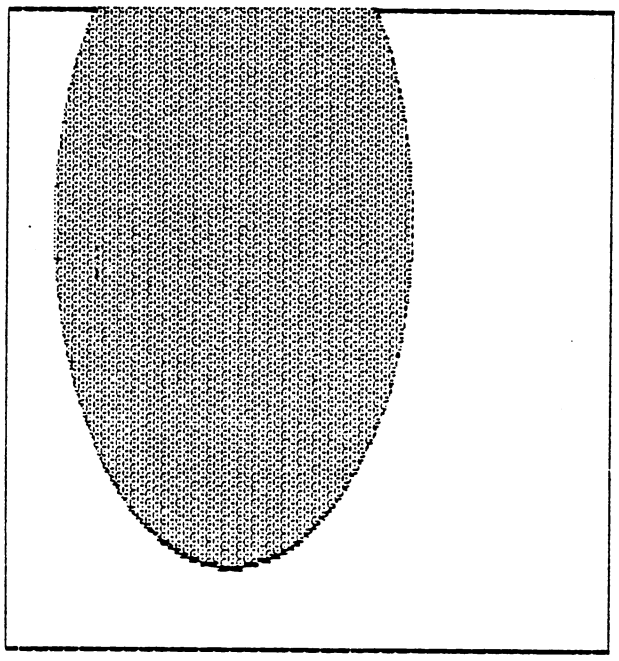
When a part to which tiling was applied is painted again in a different pattern, it takes time and sometimes leaves unpainted areas. Also, if a horizontal line in the area color exists in the boundary, that part may be left unpainted.

REFERENCE

COLOR, PALETTE, VIEW

SAMPLE PROGRAM

```
Ready
LIST

100 'PAINT statement
110 CLS:SCREEN 1:WIDTH 80
120 LINE(0,0)-(320,150),PSET,7,B
130 CIRCLE(120,50),95,2,,,.85
131 T$=CHR$(&HAA)+CHR$(&H55)+CHR$(0)
132 PAINT(120,50),T$,2
160 END

Ready
```



Line 120 draws a rectangle in white, Line 130 draws an oval in red in the rectangle, and Line 132 paints the boundary of the oval red and inside the oval with vertical blue and purple lines.

PALETTE

FUNCTION	Allocates a color to a palette.
FORMAT	PALETTE [<palette No.>, <color code>]
STATEMENT EXAMPLE	PALETTE 5, 12

TERMINOLOGY

<palette No.> : Numeric expression in a range of 0 ~ 7, that specifies a Palette No.
<color code> : Numeric expression in a range of 0 ~ 15, that specifies a color.

EXPLANATION

This statement allocates the color specified by the <color code> to the palette specified by the <palette No.>. The color specification system adopted by S1 BASIC is not direct specification of the <color code>, but a <palette No.> having the color information is specified. By this method, the flexibility of color specification and changes have been greatly improved.
The <palette No.> can be specified with a value in the range 0 ~ 7. By allocating a <color code> in the range 0 ~ 15 to each palette, up to eight colors can be used at a time.
To specify the screen color, specify this <palette No.>. For example, supposing that, if "3" is specified as the <palette No.> when a color code "5" has been allocated to the <palette No.>, the color actually displayed will be cyan (Color code 5) and not magenta (Color code 3). However, since "0" is specified as the Palette No. of the background color, if a color code for other than black is specified as Palette No. 0, the background color will change to the specified color.
The relationship between the <color code> and <palette No.> when BASIC starts is as shown below.

Palette code	Color code	Color
0	0	Black
	1	Dark blue
	2	Dark red
	3	Dark magenta (purple)
	4	Dark green
	5	Dark cyan (aquamarine)
	6	Dark yellow
	7	Dull white
	8	Black
1	9	Bright blue
2	10	Bright red
3	11	Bright magenta (light purple)
4	12	Bright green
5	13	Bright cyan (aquamarine)
6	14	Bright yellow
7	15	Bright white

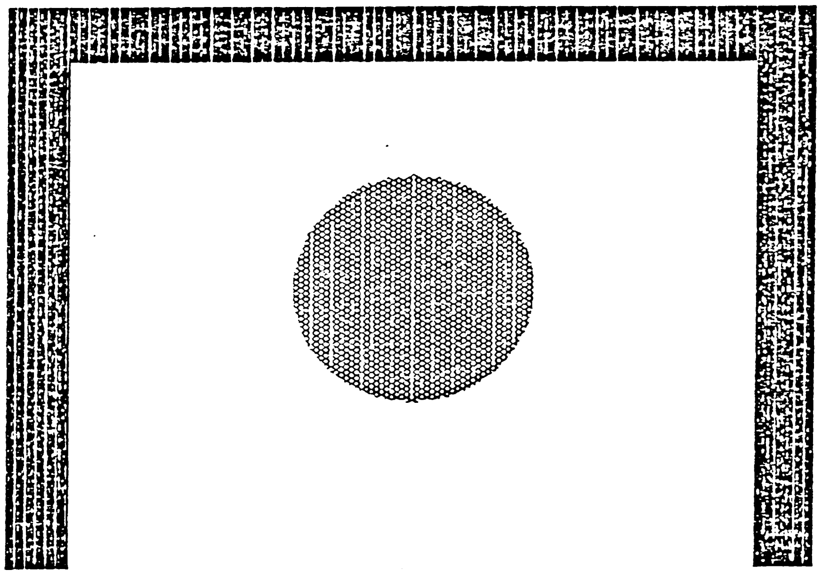
To obtain the correspondence of the Color code and Palette No. at the start of BASIC using a PALETTE statement, just specify PALETTE.
Note that the same <color code> can be allocated to two or more of <palette Nos.>, but the reverse is not possible.

REFERENCE

COLOR

SAMPLE PROGRAM

```
LIST

100 'PALETTE statement
110 CLS:SCREEN 1
120 LINE(50,20)-(590,200),PSET,7,BF
130 CIRCLE(320,100),95,2
140 PAINT(320,100),2,2
150 FOR I=0 TO 7
160 PALETTE I,15-I
170 NEXT
180 LCOPY 5
190 PALETTE
200 A=INT(ABS(RND*16))
210 IF (A)0 AND A(7) THEN 220 ELSE 200
220 PALETTE 2,A
230 FOR K=1 TO 550:NEXT K
235 PAINT(320,100),2,2:GOTO 200
240 END

Ready
```



PLAY

PURPOSE	Playing music consisting of chords notes of up to six.
FORMAT	1 PLAY "1 ch music data" [, "2 ch music data".... [, "6 ch music data"]] 2 Abbreviated form when there is no data in 2 ~ 5 ch: PLAY "1ch music data", "".... , "6 ch music data"
STATEMENT EXAMPLE	PLAY "CDEFGAB", "O5CDEFGAB"

EXPLANATION

This statement plays music using music data. The music data is a language that specifies the interval, octave, tempo, sound length and rests. When music data is specified by delimiting each of its contents with a comma, up to six notes can be played as a chord. Also, A$ = "music data" can be handled as character strings. Input all music data in capital letters, except for semi-tone b.

(1) Interval (A ~ G)

Scale	Specification method	
	White note	Black note
do	C	C♯, C+, Db, D−
re	D	D♯, D+, Eb, E−
mi	E	——
fa	F	F♯, F+, Gb, G−
so	G	G♯, G+, Ab, A−
la	A	A♯, A+, Bb, B−
ti	B	——

(Note) "b" must be the small letter.

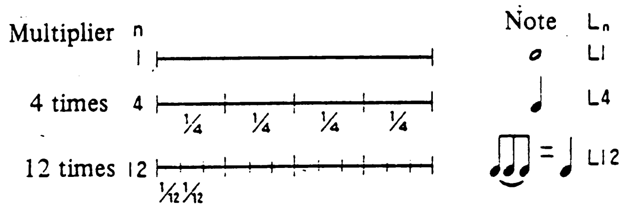
(2) Octabe (On)
Specify an octave, with "n" in an integer in the range 1 ~ 8. The "la" sound (A) reference frequency 440 Hz is O4. The initial set value is O4.

(3) Direct specification of interval (Nn)
The octave of intervals A ~ G can be specified by an On command, but it can also be specified by "N". "n" of "Nn" is an integer in the range 1 ~ 96.

Interval \ Octave		1	2	3	4	5	6	7	8
do	C	1	13	25	37	49	61	73	85
do ♯	C ♯	2	14	26	38	50	62	74	86
re	D	3	15	27	39	51	63	75	87
re ♯	D ♯	4	16	28	40	52	64	76	88
mi	E	5	17	29	41	53	65	77	89
fa	F	6	18	30	42	54	66	78	90
fa ♯	F ♯	7	19	31	43	55	67	79	91
so	G	8	20	32	44	56	68	80	92
so ♯	G ♯	9	21	33	45	57	69	81	93
la	A	10	22	34	46	58	70	82	94
la ♯	A ♯	11	23	35	47	59	71	83	95
ti	B	12	24	36	48	60	72	84	96

(4) Sound length (Ln)

This specifies the sound length. "n" is an integer in the range 1 ~ 64, and it has the value of the reciprocal of the sound fraction with "1" being the maximum. When this command is specified, it remains effective until a different sound length is specified. Specify L12 for a demi-semi-quaver.



The initial set value is L4.

(5) Rest (Rn)

This specifies a rest. "n" is an integer in the range 1 ~ 64.

L2C.

Sound length			Rest		
Note	Content	Ln	Note	Content	Rn
𝅝	Whole note	L1	𝄻	Whole rest	R1
𝅗𝅥.	Dotted half note	L2 <interval>	𝄼.	Dotted half rest	R2.
𝅗𝅥	Half note	L2	𝄼	Half rest	R2
𝅘𝅥.	Dotted quarter note	L4 <interval>	𝄽.	Dotted quarter rest	R4.
𝅘𝅥	Quarter note	L4	𝄽	Quarter rest	R4
𝅘𝅥𝅮.	Dotted eighth note	L8 <interval>	𝄾.	Dotted eighth rest	R8.
𝅘𝅥𝅮	Eight note	L8	𝄾	Eighth rest	R8
𝅘𝅥𝅯.	Dotted sixteenth note	L16<interval>	𝄿.	Dotted sixteenth rest	R16.
𝅘𝅥𝅯	Sixteenth note	L16	𝄿	Sixteenth rest	R16
𝅘𝅥𝅰.	Dotted thirty-second note	L32<interval>	𝅀.	Dotted thirty-second rest	R32.
𝅘𝅥𝅰	Thirty-second note	L32	𝅀	Thirty-second rest	R32
𝅘𝅥𝅱.	Dotted sixty-fourth note	L64<interval>	𝅁.	Dotted sixty-fourth rest	R64.
𝅘𝅥𝅱	Sixty-fourth note	L64	𝅁	Sixty-fourth rest	R64

Double dots can be specified for a dotted note or a rest.
When the next sound length is not specified after a dotted note or after a rest, the initial sound length or rest is continued and the dot part is not continued.

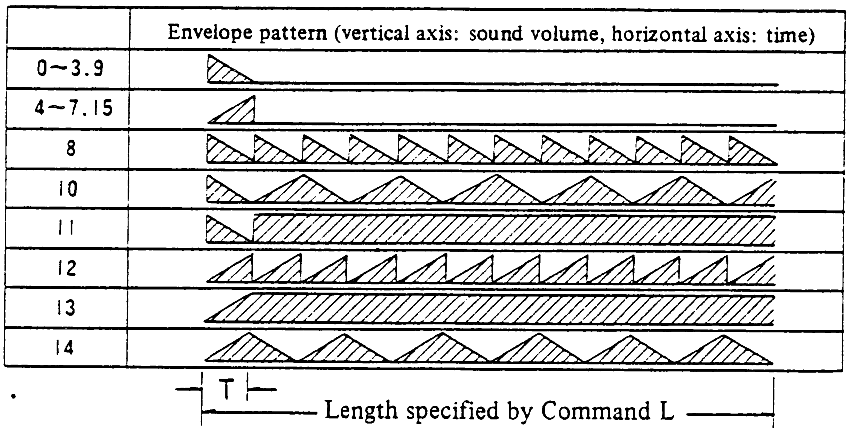
(6) Tempo (Tn)
This specifies the tempo. "n" is an integer in the range 32 ~ 255.
"n" is the speed value at which a quarter note is counted four times a minute.
T100 is equivalent to ♩ = 100.
Once this command is specified, it remains effective until another specification is given.
The initial value is T120.

(7) Sound volume (Vn)
This specifies the sound volume of a part. "n" is an integer in the range 0 ~ 15. "15" is the maximum volume and "0" emits no sound.
Once this command is specified, it remains effective until the next V or S command is specified. The initial value is V8.

(8) Envelope pattern (Sn)
This specifies an envelope. "n" is an integer in the range 0 ~ 15. Specification of an envelope enables the computer to emit a tremolo or sound attenuation, but it makes the sound volume command (Vn) invalid. To reset this specification, specify a sound volume with a sound volume (Sn) command.

(9) Envelope frequency (Mn)
This command is used in a set with an Sn command. (Be sure to specify Mn and Sn concurrently.) "n" is an integer in the range 0 ~ 65535 to specify an envelope frequency. The relational expression between period T and command M is as shown below:

$$T = \frac{256 \cdot n}{f} \text{ (sec)} \qquad n = 0 \sim 65535 \qquad f = 1.008 \text{ MHz}$$


Envelope pattern (vertical axis: sound volume, horizontal axis: time)
0~3.9
4~7.15
8
10
11
12
13
14
T
Length specified by Command L

(10) Other specifications

- MF (Music foreground)

  When MF is specified, the next PLAY statement or any BASIC statement is not executed until immediately before (the last one note or rest) terminating the PLAY statement being executing currently.

- MB (Music background)

  This specification executes the next BASIC statement while executing a PLAY statement. This instruction enables the computer to play music while executing a BASIC statement. The initial setting is MB.

- MN (Music normal)

  This specification causes 7/8 output and 1/8 rest of the sound length specified by Ln. The initial setting is MN.

- ML (Music legato)

  This specification causes the computer to output the sound length specified by Ln as it is.

- MS (Music staccato)

  This specification causes 3/4 output and 1/4 rest of the sound length specified by Ln.

Function list

		Range of "n"	Initial setting	Continuity after setting	Must be set each time
Interval	A~G				○
Octave	On	1 ~ 8	O4	○	
Special interval	Nn	1 ~ 96			○
Sound length	Ln	1 ~ 64	L4	○	
Dot	Rn	1 ~ 64			○
Tempo	Tn	32 ~ 255	T120	○	
Sound volume	Vn	0 ~ 15	V8	○	
Envelope pattern	Sn	0 ~ 15		○	
Envelope frequency	Mn			○	
Music foreground	MF			○ (Note)	
Music background	MB		○	○ (Note)	
Music normal	MN		○	○	
Music legato	ML			○	
Music staccato	MS			○	

(Note) When one PLAY statement contains two or more MF and MB, MF or MB expressions, only that written last is effective.

(Example) PLAY "MBAAA", "CCCMF" This is played by MF.

NOTE

(1) A PLAY command is terminated in the following cases:

1 When CTRL + C, or BREAK are pressed in a state other than that of waiting for the input of a BASIC command.

2 When CTRL + D are pressed.

3 When a RUN command is executed.

4 When a NEW command is executed.

5 When a NEW ON command is executed.

(2) A SOUND command (specification of SOUND <register No.>, <data>, <SG No.>) during background music, sometimes, the music becomes inaudible or it is mixed with noise because of the rewritting of the PSG register by the SOUND command. In such a case, the PSG is set to the initial state by one of the conditions listed for Item (1).

(3) A PSG option card is needed to output sound of four or more notes.

REFERENCE

SOUND

SAMPLE PROGRAM

```
LIST

10 PLAY"T90V10","T90V8","T90V8"
20 PLAY"O4R8MLL8AMNB6MSO5DMLL8CO4L16B6MNAMSL8GCMNL8FGAL16B6A",
   "O4MLL2CMNL2CL8FEFL16GF","O3MLL2FL2FL2F"
30 PLAY"MLL2G","MLL2E","MLO4L2C

Ready
RUN

Ready
```

Execution of this program produces of music with chords of three notes.

POINT

FUNCTION	Changes the LP. (Refer to 1.10 Coordinate systems.)
FORMAT	POINT (<Wx, Wy>)
STATEMENT EXAMPLE	POINT (100, 50)

EXPLANATION

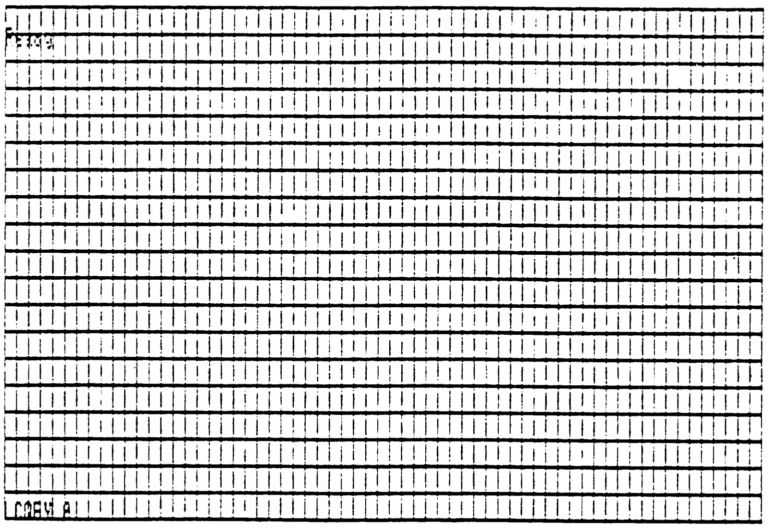
BASIC always keeps in memory the coordinates (LP, refer to 1.10 Coordinate systems) at which the last graphic operation was conducted. A POINT statement changes the coordinates.
Specify world coordinates as the coordinates.

REFERENCE

LINE, CONNECT

SAMPLE PROGRAM

```
Ready
LIST

100 'POINT statement
110 CLS:SCREEN 1:WIDTH 80
112 FOR X=0 TO 630 STEP 10
114 FOR Y=0 TO 190 STEP 10
120 GOSUB 500
130 LINE -(630,190),PSET,6,B
132 NEXT
134 NEXT
140 END
500 '
530 POINT(X,Y)
540 RETURN

Ready
```



Line 130 draws a box while changing the starting point by Line 530.

POKE

FUNCTION — Writes data to a specified memory address.

FORMAT — POKE <memory address>, <data>

STATEMENT EXAMPLE — POKE &H6FFF, 18

EXPLANATION

This statement writes <data> (one byte) to the specified <memory address>. A numeric expression is permitted in these specifications, but the specification of <memory address> must be a hexadecimal number in the range 0 ~ 65535 and the specification of <data> must be in the range 0 ~ 255.
In S1 BASIC, the data is written to the machine language area specified by the DEF MAP statement. When necessary, change the machine language area by inputting a DEF MAP statement.

REFERENCE

PEEK function and DEF MAP

SAMPLE PROGRAM

```
Ready
LIST

100 'POKE statement
110 CLEAR ,&H8000
120 FOR I%=0 TO 4
130 READ J%
140 POKE I%+&H9000,J%
150 NEXT I%
160 END
170 DATA &HFF, &HFF, &HFF, &HFF, &HFF

Ready
RUN

Ready
```

Lines 120 through 150 write &HFF to five bytes from &H9000.

PRESET

FUNCTION Erases the point of the specified coordinates.

FORMAT PRESET (<Wx>, <Wy>)

STATEMENT EXAMPLE PRESET (50, 100)

EXPLANATION

This statement erases the point of the specified position on the screen. The erased point is colored by the background color specified by a COLOR statement.

REFERENCE

PSET

SAMPLE PROGRAM

```
Ready
LIST

100 'PRESET statement
110 CLS:SCREEN 1
120 LINE(0,100)-(639,100),PSET,4
130 FOR X=0 TO 639
140 PRESET (X,100)
150 NEXT X
160 END

Ready
```

Line 110 erases the screen display. The FOR ~ NEXT loop on Lines 130 through 150 erases the green line while changing the x coordinate value.

PRINT

FUNCTION Outputs an expression value on the display.

FORMAT

```
                         | ; |
PRINT [<expression> [    | , |  <expression>] ...
                         | ␣ |
  | ; |
[ | , | ]]
```

(The abbreviated form is ?.)

Note) 「␣」 means a space.

STATEMENT EXAMPLE PRINT "BASIC"

EXPLANATION

This statement displays the value (including an independent constant and variable) of <expression> (including a character expression) on the screen. If nothing is described after PRINT, only line feed is performed.

- When a semi-colon 「;」 or space 「␣」 is written after the <expression>, the next <expression> is displayed immediately after the preceding <expression>, but the semi-colon or space can be omitted except when the first character of the next <expression> is a code or the next <expression> is a variable name without an attribute character and it is followed by alphanumerics. If the end of a PRINT statement is a semi-colon 「;」, until a new PRINT or LOCATE statement is executed, the cursor does not move (is not displayed).
- One line is divided into groups consisting of 14 characters, and when a comma 「,」 is written after the <expression>, the next <expression> is displayed in the 14-character area next to the 14-character area where the last character of the preceding <expression> was displayed. Since one line is divided into 14-character groups, the last area of the line is shorter than the 14-character area size. There is no display in this area, and the part that should go in this area is displayed from the start of the next line.
- Also, regardless of the delimiting sign, if a numeric value or character string that is displayed in the middle of a line cannot be completed in the line and must be continued on the next line, the display of the numeric value or character string is automatically shifted to the start of the next line.

NOTE

When the <expression> is a numeric value, if a space ⌜␣⌟ is used as a delimiter between an <expression> and another <expression>, the data may sometimes change. Do not use a space ⌜␣⌟ when the <expression> is a numeric value.

REFERENCE

PRINT USING, LPRINT, LPRINT USING

SAMPLE PROGRAM

```
Ready
LIST

100 'PRINT statement
110 A=12345
120 PRINT "A",A
130 PRINT "A";A
140 PRINT "A" A
150 A=1.24247E-09
160 PRINT "A" A
170 END

Ready
RUN
A                 12345
A 12345
A 12345
A 1.24247E-09

Ready
```

This program shows the display format when using a comma, semi-colon and space as a delimiter on Lines 120 through 140.

PRINT USING

FUNCTION Displays character strings and numeric values as shown by the format expression.

FORMAT PRINT USING "<format expression>"; <expression> [| ; , | <expression>] ... [| ; , |]

STATEMENT EXAMPLE PRINT USING "#####.#" ; 1000, 5

EXPLANATION

The output format of printed characters and numeric values is determined by the pre-determined format control characters contained in the <format expression>. These format control characters are not output to the screen as characters.
When the last character in a PRINT USING statement is a semi-colon 「;」 or comma 「,」, the cursor does not move (is not displayed) until a PRINT or LOCATE statement is executed.

Format control of color strings

- 「!」

  Outputs the only first character of the given character string.

  (Example)
```
PRINT USING "!is the top.";  "YAMA", "KAWA", "MORI"
Yis the top. Kis the top.  Mis the top.
```

- 「&」 and 「 ␣ 」 (space)

  The specification must be done in the form of "N" number of spaces 「   」 (0 ≦ N) are always sandwitched between two 「&」 signs. When this specification is given, (2 + N) characters from the start of the given character string are output and the rest of the characters are ignored.
  If the character string is shorter than the specified length, the output is done in the left justified form within the area, and the remaining part is filled by space 「   」.

  (Example)
```
PRINT USING "&    & ";"HITACHI","MB-S1 BASIC"
HITAC MB-S1
```

A character variable, Hollerith constant or character expression may be specified as the character string. Note that specifying the character string with a character array may cause erroneous processing. In such a case, assign it to a character variable before specifying it, or add 「+""」 to the character array before output and specify it as a character expression.

(Example)
```
PRINT USING "& &" ; B$ (I) +""
PRINT USING "!" ; B$ (I)+""
```

Format control of numeric value display

- 「#」 and 「.」 (period)

  「#」 shows the digit to be displayed as a numeral and 「.」 shows the position to be displayed as a decimal point. Therefore, the display is made in the fixed point form. If 「.」 is not specified, the output is shown as an integer.
  When the number of digits of a numeric value is smaller than the specified number of digits, the output is in the right justified form within the area and the higher digits are filled by spaces. "0" is output as redundant digits in fractions. Note that one 「#」 is used as a code.

  (Example)
```
PRINT USING "### ";200,-25,125,12.25
200 -25 125  12

PRINT USING "####.## ";2.1,-2.5,125.35
   2.10    -2.50   125.35
```

- 「+」 and 「−」

  Write 「+」 or 「−」 before or after a list of numeric value format control characters. However, note that 「−」 can only be written after a character string.
  When 「+」 is written, the code of a numeric value is output either before or after the numeric value. When 「−」 is written, a minus sign is output after the numeric value, only when the numeric is negative.
  Note that 「+」 or 「−」 always secures a digital position to be shown.

  (Example)
```
PRINT USING "+####.## ";200,-25.725,1250,-12.25
 +200.00    -25.73 +1250.00    -12.25

PRINT USING " ####.##- ";200,-25.725,1250,-12.25
  200.00      25.73-   1250.00      12.25-
```

- 「,」 (comma)

  Write a comma 「,」 at the left side of a period 「.」 to specify a decimal point. If a comma is written, the integer part is delimited by a comma for every three digits from the right side. Note that a digital position is used to show the comma.

  (Example)
```
PRINT USING"#########,     ";3000000,123.456,12345
 3,000,000              123          12,345

PRINT USING "#########,.## ";3562148.263,562,148.26
 3,562,148.26             562.00            148.26
```

- 「**」

  Write this at the start of a list of numeric value format control characters. This uses two digits, but when there is a blank area in the upper digits of a numeric value, the area is filled by asterisks instead of spaces.

  (Example)
```
PRINT USING "**####.## ";3750.25,1.128,565361.2
**3750.25 *****1.13 655361.00
```

■ 「¥¥」

Use this in the same way as 「**」. However, note that only one 「¥」 sign is output immediately before the numeric value. 「¥¥」 cannot be used to specify a floating point format.

(Example)
```
PRINT USING "¥¥########,  ";38000000,10000,13.25
¥38,000,000      ¥10,000          ¥13

PRINT USING "+¥¥########,  ";38000000,10000,13.25
+¥38,000,000      +¥10,000          +¥13

PRINT USING "¥¥########,-";-38000000,10000,-13.25
¥38,000,000-     ¥10,000          ¥13-
```

■ 「**¥」

Use this in the same way as 「**」 or 「¥¥」. This uses three digits on the display. If there is a blank are at the upper part of numeric, one digit is filled by 「¥」 and the rest by 「*」.

(Example)
```
PRINT USING "**¥#######,    ";5000000,10000,3.14
¥5,000,000   ***¥10,000   *********¥3
```

■ 「∧∧∧∧」

Write this after 「#」 that specifies the digit. Writing this specifies the floating point format, and 「#」 specifies the display digits of the mantissa.

(Example)
```
PRINT USING "#.####^^^^ ";12345,4.2E20,0.00125
0.1235E+05 0.4200E+21 0.1250E-02

PRINT USING "##.####^^^^ ";12345,4.2E20,0.00125
 1.2345E+04  4.2000E+20  1.2500E-03
```

NOTE

If the number of digits of a numeric value to be displayed exceeds the display area size, 「%」 is output before the numeric value. This also applies in the case of rounding numeric values by the method of regarding fractions of .5 and over as whole numbers and discarding the rest.
If a character other than the above format control characters is used in the <format expression>, it is output to the screen following the positional relationship (before or after) with a format control character.

REFERENCE

PRINT, LPRINT, LPRINT USING

SAMPLE PROGRAM

```
Ready
LIST

100 'PRINT USING statement
110 A=1:B=234:C=567890!:D=1.2
120 AM=-A:BM=-B:CM=-C:DM=-D
130 PRINT USING "######.##  ";B;BM;C;CM
140 PRINT USING "+#####.##  ";A,AM,B,BM
150 PRINT USING "######.##- ";D;DM;C;CM
160 PRINT USING "**####.##  ";A;AM;B;BM
170 PRINT USING "¥¥####.##- ";A,B,C,D
190 PRINT USING "#####,.### ";A,B,C
200 END

Ready
RUN
   234.00    -234.00  567890.00  %-567890.00
    +1.00      -1.00    +234.00    -234.00
     1.20       1.20- 567890.00  567890.00-
*****1.00  ****-1.00  ***234.00  **-234.00
    ¥1.00    ¥234.00  %¥567890.00      ¥1.20
     1.000    234.000 %567,890.000

Ready
```

Various format control characters are used in this program.

PRINT # (USING)

FUNCTION

Outputs an expression value (format control) to an output file.

FORMAT

1 When outputting to a cassette, display or printer

PRINT #<file No.> [, <expression> [| ; , ␣ | <expression> ...] [| ; , |]]

2 When outputting to the RS-232C line

PRINT #<file No.> [, <expression> [| ; , ␣ | "," | ; , ␣ | <expression> ...] [| ; , |]]

3 PRINT #<file No.>, USING "<format expression>"; <expression> [| ; , | <expression> ...] [| ; , |]

STATEMENT EXAMPLE

```
1  PRINT # 1, A ; B ; C$
2  PRINT # 1, A$ ; "," ; BS ; "," ; C
3  PRINT # 2, USING "###.##" ; X ; Y ; Z (100)
```

EXPLANATION

This statement outputs an expression value to a specified file. The file to be specified must be opened in output mode 「"O"」 in advance.
The meanings of delimiting signs 「;」, 「,」 and 「␣」 used in the PRINT # statement are the same as those used in PRINT statements.
A PRINT # USING statement is used for output to a file under format control. For the details of <format expression>, refer to the explanation of the PRINT USING statement. Also, if the last character of a PRINT # USING statement is 「;」 or 「,」, the cursor does not move (is not displayed), as in the case of the PRINT USING statement.
When outputting two or more expressions to the RS-232C line, 「","」 is placed between two expressions as shown in Format 2 .

NOTE

Cautions on output to a printer are explained in the following using an actual program as an example.

(Example)
```
10  OPEN "0", 1, "LPT0 :"
20  FOR I=1 TO 5
30  PRINT #1, I ;
40  NEXT
```

Nothing is output to the printer if this program is executed. This is because, if the last character of a PRINT # statement is 「;」 as on Line 30, the return and line feed codes to the printer are not output to the buffer and the characters remain in the buffer. The same applies to a PRINT # USING statement, and the same thing occurs if the last character of a PRINT # (USING) statement is 「,」. In such a case, execute a CLOSE statement with the computer waiting for a command to close the buffer and have the characters in the buffer output to the printer.

Add the following to this program:

```
50 PRINT #1
```

This time, when the program is executed, the return and line feed codes are output to the buffer and the characters in the buffer are output to a printer. In this state, since the file is kept in the open state, if there is an instruction to use a printer in the program after Line 50, for example, the program can be executed continuously.

When all processing of an opened file is complete, execute a CLOSE statement to close the file, and output the characters in the buffer.

When an instruction to output to a printer is given without connecting a printer, the computer waits for a signal from the printer and the program is not executed. In such a case, key in $\boxed{\text{CTRL}}$ + $\boxed{\text{D}}$ and execute a CLOSE statement as a command.

Conduct the following when computer operation is stopped during output to a printer by keying in $\boxed{\text{CTRL}}$ + $\boxed{\text{D}}$:

(1) Input a CLOSE instruction to output characters remaining in the buffer.
    CLOSE

(2) Input as follows when the characters remaining in the buffer are not to be output:
    PRINT #n, CHR$ (&H18)
    CLOSE n
    n (small N): File No. of printer
    In the above example, "n" is "1".

REFERENCE

OPEN, CLOSE, INPUT #, LINE INPUT #, PRINT #

SAMPLE PROGRAM

```
LIST

100 'PRINT #,  PRINT # USING  statement
110 OPEN "O",#1,"TEST"
120 A$="ABC":B$="DEF":C$="GHI"
130 PRINT #1,A$;",";B$;",";C$
140 CLOSE
142 PRINT"ﾃｰﾌﾟ ｦ ﾏｷﾓﾄﾞｼﾃｸﾀﾞｻｲ !!"
145 STOP
150 OPEN "I",#1,"TEST"
160 INPUT #1,X$,Y$,Z$
170 PRINT X$
180 PRINT Y$
190 PRINT Z$
200 CLOSE
210 END

Ready
RUN
ﾃｰﾌﾟ ｦ ﾏｷﾓﾄﾞｼﾃｸﾀﾞｻｲ !!

Break In 145
Ready
CONT
ABC
DEF
GHI

Ready
```

Line 110 opens an output file called "TEST". Lines 120 and 130 output character strings A$, B$ and C$. Lines 150 through 200 open the prepared file, fetch data into X$, Y$ and Z$, and display the result.

PSET

FUNCTION Sets a dot at the specified coordinates.

FORMAT PSET (<Wx, Wy>) [, <color>]

STATEMENT EXAMPLE PSET (100, 200), 3

EXPLANATION

This statement sets a dot at the specified position on the screen. The <color> is the dot color. Specify it with a Palette No. If the specification of the <color> is omitted, the character color specified by the COLOR statement immediately before is used.
When a coordinate value outside of the screen display range is specified, the dot is not displayed. However, care must be paid to this since it is effective as LP (refer to 1.10 Coordinate systems).
Specify a value in the range 0 ~ 15 for the <color code>. Refer to the explanation of the COLOR statement for the correspondence between the color and color code.

REFERENCE

PRESET

SAMPLE PROGRAM

```
Ready
LIST

100 'PSET, PRESET statement
110 CLS
120 FOR X=0 TO 639
130 PSET (X, 100), 2
140 NEXT X
150 END

Ready
```

This program draws a red horizontal line using a PSET statement.

PUT @

FUNCTION

Outputs a graphic pattern or text pattern to the screen.

FORMAT

1 PUT (<Sx, Sy>), <array name> [. [<plot option>] [, [<horizontal magnification>] [, <vertical magnification>]]]
2 PUT @ (<Sx, Sy>), <array name> [. [<plot option>] [, [<color>] [. [<horizontal magnification>] [, <vertical magnification>]]]]
3 PUT @C (<X, Y>), <array name>

STATEMENT EXAMPLE

1 PUT (100, 50), A, PSET, 2, –2
2 PUT @ (X, Y), B, XOR, 5, –2, –2
3 PUT @C (10, 10), C

EXPLANATION

This statement displays the graphic or character pattern made ready by the PUT @ statement at the specified pattern on the screen.
In the Format 2 , PUT @ statement, specify a Palette No. with the dot pattern color to be displayed as the <color>. If this specification is omitted, the character color specified by a COLOR statement is used.
The display methods are shown in the following table.

Item			Format 1	Format 2	Format 3
1	Function outline		Displays the color graphic pattern in the array on the screen.	Displays the dot pattern in the array in the specified color.	Displays the character pattern in the array on the screen.
2	Coordinate system		Screen coordinates	Screen coordinates	Text coordinates
3	Array Data format		Refer to GET @	Refer to GET @	Refer to GET @
4	Relationship between plot option and display method	Omitted	Displays the graphic pattern in the array on the screen without any change.	Clear the background first and then plots the places where the bits of the dot pattern in the array are set to "1" using the specified color.	Displays the character pattern in the array on the screen without any change.
		NOT	Inverts all bits of the graphic pattern in the array and displays the result. Therefore, the display is in the complementary color of the pattern in the array.	Draws the background in the specified color first, and then resets the places where the bits of the dot pattern in the array are set to "1" to the background color. That is, the display is made in the reversed state.	
		PSET	Same as the case of omission.	Plots the places where the bits of the dot pattern in the array are set to "1" with the specified color. The background is not changed. Therefore, the display is made over the pattern displayed at present.	
		PRESET	Same as specifying NOT.	Resets the places where the bits of the dot pattern in the array are set to "1" to the background color. No change takes place where the bits are "0". Therefore, the display is made over the pattern displayed at present.	
		AND OR XOR	Conducts a logical operation on the graphic pattern in the array and graphic pattern displayed at present, and displays the result on the screen.	Conducts a logical operation on the specified Palette No. with regard to the places where the bits of the dot pattern in the array are set to "1" and the Palette No. of dots displayed on the screen at present, and plots the dots using the resulting palette No. The places where the bits are "0" remain unchanged.	
			Specification of XOR among the three returns the graphic pattern on the screen to the initial state by applying two PUT commands to the same array. This function is especially useful in animation pictures to move the graphic pattern in the array without destroying the graphic pattern on the screen that is used as the background.		

Specify the magnifications in the horizontal and vertical directions of the character to be displayed for the <horizontal magnification> and vertical magnification in Formats 1 and 2 using an integer. If this specification is omitted, the default value is "1". When a negative value is specified for the <horizontal magnification>, the character is inverted in the horizontal direction, and if a negative value is specified for the <vertical magnification>, the character is inverted in the vertical direction. When a negative value is specified for the both, the character is inverted in both the horizontal and vertical directions on the screen. The magnification is expressed in an absolute value.
When Format 3 is used, the output state on the screen (whether the priority of overlapping is given to a text or graphic pattern) can be changed by changing the attribute of character read by GET@C.

REFERENCE

GET@ and SCREEN function

R

RANDOMIZE

FUNCTION — Specifies a random number.

FORMAT — RANDOMIZE [<argument>]
(The abbreviated form is RNDM.)

STATEMENT EXAMPLE — RANDOMIZE (TIME/3)

EXPLANATION

Random numbers obtainable by a RND function are in actuality pseudo random numbers generated based on a value (seed of random numbers) that is given according to a set expression. Therefore, no matter how many times a program is executed, the resulting random number system is the same. A RANDOMIZE statement is written at the start of a program in order to avoid this (if inserted in the middle of a program, it stipulates the contents after the inserted place).
When the <argument> specification is omitted, the screen displays the following for each RUN command.

RND Seed (−32768 to 32767)?

This requests the input of a seed to determine the random number system. If a new numeric that has not been used is input, a random number system that is different from the preceding one is obtained. When the TIME function is used for the <argument>, a new random number system is available for each run.

REFERENCE

RND function

SAMPLE PROGRAM

```
Ready
LIST

100 'RANDOMIZE statement
110 RANDOMIZE(TIME/3)
120 FOR I=0 TO 4
130 PRINT CINT(100*RND);
140 NEXT
150 END

Ready
RUN
 57  40  96  10  11
Ready
RUN
 92  13  70  28  6
Ready
```

When a RUN instruction is executed, a series of random numbers are obtained according to the value of TIME executed by Line 110.

READ

FUNCTION Reads data defined by a DATA statement into variables.

FORMAT READ <variable name> [, <variable name>] ...

STATEMENT EXAMPLE READ !, B$, C (100)

EXPLANATION

This statement reads the constants specified in a DATA statement and assigns them into variables. A READ statement must always be used in a set with a DATA statement. A READ statement assigns data from the DATA statement into variables with a 1:1 correspondence. When a variable to be assigned is a numeric variable, the corresponding data must also be a numeric constant.

One READ statement can read data from one or more DATA statements. Also, two or more READ statements can read data from one DATA statement. In either case, the DATA statement is read in the order of the lowest Line No. to the highest Line No., from the start of each data line, unless a RESTORE statement is executed. The following shows an example of reading data from DATA statements.

(Example)

```
Ready
LIST

10 DATA A,1,B,C,2,D,E
20 READ A$,I,B$:PRINT "A$=";A$,"I=";I,"B$=";B$
30 IF I=1 GOTO 20
40 READ C$,D$,J,K
50 PRINT "C$=";C$,"D$=";D$
60 DATA F,3
70 PRINT "J=";J,"K=";K
80 DATA 4

Ready
RUN
A$=A          I= 1          B$=B
A$=C          I= 2          B$=D
C$=E          D$=F
J= 3          K= 4

Ready
```

DATA statements are written on Lines 10 and 60 and READ statements are written on Lines 20 and 40.

The READ statement on Line 20 corresponds with three variables and it reads three data items from the start of Line 20. At this time, since "1" is read into "I", the READ statement of Line 20 is executed again and the next three data items are read. At this time, "2" is read into "I" and the execution moves to Line 40.

The READ statement on Line 40 needs four data items. Since Line 10 has one data item that has not been read, this data is read first and then the data on Line 60 and the data on Line 80 are read sequentially.
If the number of variables described in a READ statement is greater than the number of data items described in a series of DATA statements, an "Out of Data" error occurs.
If too many data items are described in DATA statements, the find parts are ignored.
Data items of DATA statements can be reread by a RESTORE statement.

REFERENCE

DATA, RESTORE

SAMPLE PROGRAM

```
Ready
LIST

100 'READ statement
110 FOR I=0 TO 4
120 READ NAME$(I)
130 READ SCORE(I)
140 NEXT I
150 RESTORE 230
160 FOR I=0 TO 4
170 READ HANDI
180 NET(I)=SCORE(I)-HANDI
190 PRINT NAME$(I),NET(I)
200 NEXT
210 DATA HITACHI,14,ﾔﾏﾀﾞ,19,ｽｽﾞｷ,8
220 DATA YAMAMOTO,3,SATO,-1,KANAYA,8
230 DATA 12,16,10,2,0,0

Ready


RUN
HITACHI        2
ﾔﾏﾀﾞ           3
ｽｽﾞｷ          -2
YAMAMOTO       1
SATO          -1

Ready
```

The READ statements on Lines 120, 130 and 170 input character strings and numerics, and Line 190 prints out each data.

REM (Remark)

FUNCTION Adds a comment in a program.

FORMAT REM <character string> (The abbreviated form is 「'」.)

EXPLANATION

This statement adds a comment in program. A REM statement is a non-executed statement and it is ignored during program execution but it is output as it is when a program list is output.
The execution can be branched to a REM statement by a GOTO or GOSUB statement and the execution is conducted from the first executable statement after the REM statement. An execution statement described after a REM statement is not executed even if a colon 「:」 is used after the REM statement.

NOTE

In a different way from other abbreviated forms, the abbreviated form of a REM statement, 「'」, is not converted to 「REM」 and it remains as it is.

SAMPLE PROGRAM

```
Ready
LIST

100 'REM statement
110 A=10
120 REM A=0
130 PRINT A
140 END

Ready
RUN
 10

Ready
```

Even when BASIC reads REM A = 0 on Line 120, A = 0 is not performed, and Line 130 prints "10".

RENUM command (Renumber)

FUNCTION	Renumbers program lines.
FORMAT	RENUM [<new line No.>] [, [<old line No.>] [, <increment>]]
STATEMENT EXAMPLE	RENUM 100, 50, 50

EXPLANATION

This statement renumbers program lines after the line specified by the <old line No.> to line numbers starting with the specified <new line No.> and increments the line Nos. by the specified <increment>. All line numbers in the program for GOTO, GOSUB, IF, ON ~ GOTO and ON ~ GOSUB statements and ERL function are also renumbered. If a line No. in these statements does not exists in the program, an error message ⌈Undefined Line Number <erroneous line No.> in <line No. where GOTO or other statement is described>⌋ is output and the erroneous line No. remains as it is. This is because, if the erroneous line number becomes the same as one of the new line numbers, the correct branch destination cannot be specified. Immediately correct the program.
Generation of line numbers over 64000 by a RENUM statement is not possible.
When the specification of <new line No.> or <increment> is omitted, "10" is used. If the specification of <old line No.> is omitted, renumbering starts from the lowest line No. of the present program.
If a RENUM statement appears in a program, the statement is executed and the computer is set to wait for a command.

NOTE

A RENUM command cannot be used to change the program line order. (For example, on a program having three lines numbered 10, 20 and 30, an instruction such as ⌈REM 15, 30⌋ cannot be used since old line No. 30 comes before Line 20.

SAMPLE PROGRAM

```
Ready
LIST

10 'RENUM command
15 ON ERROR GOTO 483
110 CLS
115 GOSUB 295
130 IF C<32 THEN GOTO 115
135 PRINT CHR$(C),
140 GOTO 115
295 '---------- Subroutine ----------
305 PRINT "Subroutine "
307 C=C+1
310 RETURN
483 '--------- error routine ----------
502 IF ERL<>135 THEN ON ERROR GOTO 0
503 PRINT ERR,ERL
550 RESUME 140

Ready
```
(1)

```
RENUM 100

Ready
LIST

100 'RENUM command
110 ON ERROR GOTO 210
120 CLS
130 GOSUB 170
140 IF C<32 THEN GOTO 130
150 PRINT CHR$(C),
160 GOTO 130
170 '---------- Subroutine ----------
180 PRINT "Subroutine "
190 C=C+1
200 RETURN
210 '--------- error routine ----------
220 IF ERL<>150 THEN ON ERROR GOTO 0
230 PRINT ERR,ERL
240 RESUME 160

Ready
```
(2)

This is an example of describing RENUM 100 to Program (1). After renumbering, the line Nos. start with 100 and increase by 10. The line numbers of ON ERROR GOTO, GOTO and GOSUB statements are also renumbered. Note that the numeric (line No.) that makes a pair with the ERL statement of Line 220 of Program (2) is also renumbered.

RESTORE

FUNCTION | Specifies the line to start reading data from a DATA statement.

FORMAT | RESTORE [<line No.>]

STATEMENT EXAMPLE | RESTORE 5000

EXPLANATION

This statement is used to read data of the same group for a number of times. If a RESTORE statement is executed without the specification for the <line No.>, a READ statement after the RESTORE statement reads data from the first DATA statement of the program. When the <line No.> is specified, a READ statement after the RESTORE statement reads data from the first DATA statement after the line specified by the <line No.>. The line specified by the <line No.> needs not be a DATA statement.

REFERENCE

DATA, READ

SAMPLE PROGRAM

```
Ready
LIST

100 'RESTORE statement
110 FOR I=1 TO 2
120  FOR J=1 TO 2
130   READ A:PRINT A,
140  NEXT J
150  RESTORE
160 NEXT I
170 END
180 DATA 100,200

Ready
RUN
 100           200           100           200
Ready
```

Line 180 defines two data items, but by using a RESTORE statement on Line 150, READ is conducted four times.

RESUME

FUNCTION	Returns from an error processing routine.
FORMAT	1 RESUME [<line No.>] 2 RESUME NEXT
STATEMENT EXAMPLE	RESUME 500

EXPLANATION

This statement restarts the execution of the main program after an error processing program has been executed. There are three formats to be selected depending on the place to restart execution.

RESUME	Execution restarts at the statement where the error occurred.
RESUME NEXT	Execution restarts at the statement next to the statement where the error occurred.
RESUME <line No.>	Execution restarts at the line specified by the <line No.>.

A RESUME statement indicates the end of an error processing program, and it must be always described except for the case of not returning to the main program.

REFERENCE

ERROR, ON ERROR GOTO

SAMPLE PROGRAM

```
Ready
LIST

100 'RESUME statement
110 ON ERROR GOTO 150
120 PLINT
130 ERROR 26
140 END
150 IF ERR=2 THEN PRINT
    "   syntax error  in";ERL:RESUME NEXT
160 IF ERR=26 THEN PRINT
    "   error 26 in";ERL:RESUME 190
170 PRINT "   other error"
180 ON ERROR GOTO 0
190 LOCATE

Ready
RUN
   syntax error  in 120
   error 26 in 130
   other error
Missing Operand In 190
Ready
```

Line 110 defines the line No. of the error processing routine.
After program execution start, if an error occurs, the execution jumps to Line 150, the line No. is displayed, and the statement next to the statement where the error occurred is executed. Note that RESUME is conducted to Line 130 by the RESUME NEXT statement.

RUN command

FUNCTION (Loads) and executes a program.

FORMAT

1 RUN [<line No.>] (The abbreviated form is R.)

2 RUN "[CAS0:] [<file name>]" [,R]

STATEMENT EXAMPLE

1 RUN 500

2 RUN "CAS0: TEST"

EXPLANATION

1 When the line number is omitted, the program execution starts at the lowest number line. When the line number is specified, execution starts at the specified line. Prior to starting a program, all numeric variables are initialized to "0" and all character variables to null strings. Also, all open files are closed.

2 When the file name is specified, the specified file on the cassette tape is read into memory and is immediately executed. When the specified file is found, the program in the memory is erased. The operations prior to starting the program execution are the same as Format 1 . However, open files are not closed when option R is specified.

When CTRL + D are input during cassette recorder operation, the operation is temporarily stopped and the computer is set to wait for a command. At this time, since the file is kept in the open state because of the execution of the RUN command, input a CLOSE statement via the keyboard.

This command can be used in a program.

REFERENCE

STOP, LOAD, MERGE

SAMPLE PROGRAM

```
Ready
LIST

100 'RUN command
110 A=10
120 PRINT A
130 END

Ready
RUN
 10

Ready
```

The RUN command starts the program. When RUN ↵ is input from the command level, the program is executed and "10" is displayed.

SAVE command

FUNCTION Records programs on cassette tape.

FORMAT SAVE "[CAS0;] <file name>" [, $\left|\begin{matrix}\text{A}\\\text{P}\end{matrix}\right|$]
(The abbreviated form is SA.)

STATEMENT EXAMPLE SAVE "CAS0: TEST", A

EXPLANATION

This statement records programs on cassette tape with their file names.
When option A is specified, the programs are saved in the character form (ASCII save). Although the character form requires a long time for saving and a long length of the tape to record, it can be loaded without almost any changes even the version of BASIC is changed. All program files used by MERGE and program files to be input by a LINE INPUT # statement must have been saved in character form.
If the option item is not specified, the program is output in the internal expression type (interpreter type). The internal expression type has merits of quick saving and loading and a shorter tape length used for recording, but it cannot be loaded as is when the version of BASIC is changed.
When P (Protect) is specified as the option item and a program recorded on cassette tape is loaded, the only applicable command is RUN, that is, list output or program editing cannot be done.
When a SAVE (ASCII SAVE) command is executed during program execution, the computer is set to wait for a command after executing the SAVE command.

NOTE

When option P is specified and a program recorded on cassette tape is loaded, no operation other than RUN is possible. Since there is no command to reset this state, care must be taken. Also, to save a program thus saved, "P" must be specified again. When option P is specified and the program recorded on cassette tape is loaded, a NEW or NEW ON command must be executed prior to inputting a separate new program from the keyboard.

REFERENCE

LOAD, MERGE

SAMPLE PROGRAM

```
Ready
SAVE"TEST"

Ready
NEW

Ready
LOAD"TEST"

Searching
Found: TEST

Ready
LIST

100 'SAVE command
110 PRINT "ABC"
120 END

Ready
```

This program saves a program file first. When a NEW command is executed and a file called "TEST" is loaded, the program is displayed when a LIST instruction is given.

SAVEM command

FUNCTION	Records a machine language program on a cassette tape.
FORMAT	SAVE "[CAS0:] <file name>", <start address>, <last address>, <starting address>
STATEMENT EXAMPLE	SAVEM "SOUND", &H7D00, &H7FFF, &H7D00

EXPLANATION

This statement records a machine language program from the <start address> to the <last address> with a specified <file name>.
The <starting address> specifies the address to start execution of the machine language program at the time of LOADM.

NOTE

When a character is specified for the <starting address>, the machine language program may not be recorded on the cassette tape correctly. Describe the <starting address> exactly.

REFERENCE

LOADM, DEE MAP

SAMPLE PROGRAM

```
LIST

100 'SAVEM command
110 CLEAR 300, &HC000
120 SAVEM"TEST", &HC000, &HC100, &HC000
130 PRINT "MOTOR MB-S1 PROGRAM"
140 END

Ready
```

SCREEN

FUNCTION	Specifies a screen mode and page.
FORMAT	SCREEN [<fineness mode•] [,[<page>] [,<mode>]]
STATEMENT EXAMPLE	SCREEN 1,,0

EXPLANATION

The <fineness mode> determines the screen mode. Specify one of the following:

Omission	:	The current fineness is held.
0	:	320 x 200 dots
1	:	640 x 200 dots (at the initial stage)

The <page> specification is effective when the fineness mode is 320 x 200 dots, and it specifies the page (drawing page) that is actually read/written by an instruction like LINE, CIRCLE, etc., for the graphic VRAM of Page 2 that can be used with this fineness, and the page (display page) that is actually shown on the screen. The specification methods are shown below.

Specification of <page>	Display page	Drawing page
0	0	0
1	1	1
2	0	1
3	1	0

When the specification of <page> is omitted, the current Page No. is used.
Specify one of the following for the <mode>.

<mode>	Contents
0	Interlace mode. Text screen scroll.
1	Interlace mode. Text screen scroll (at the initial stage).
2	Interlace mode. Scroll of both text and graphic screens.
3	Interlace mode. Scroll of both text and graphic screens.

NOTE &H FE24 register - see "CURTAIN"

When the <fineness> is specified in a SCREEN statement, the following processing is conducted:

(1) Screen of all pages is erased.
(2) The cursor mode is initialized.
(3) Sets Display page = Drawing page = Page 0 (when the <Page No.> specification is omitted)
(4) Set the window and view port back to the initial state.
(5) Sets LP (refer to 1.10 Coordinate systems) to the upper left of the view port.
(6) Sets the display position to the upper left of view port when the coordinates specification is omitted in a KANJI statement.

NOTE

No parameters of a SCREEN statement can be omitted.

REFERENCE

WIDTH

SAMPLE PROGRAM

```
LIST

100 'SCREEN statement
110 SCREEN 0,0,0
120 LINE (80,80)-(619,199),PSET,1,BF
130 SCREEN' ,1,0
140 LINE (10,10)-(400,150),PSET,2,BF
150 FOR I=0 TO 10
160   SCREEN  ,0,0
170   FOR J=0 TO 500: NEXT
180   SCREEN  ,1,0
190   FOR J=0 TO 500: NEXT
200 NEXT

Ready
```

Line 110 specifies "0" for the drawing page and Line 120 draws a box.
Line 130 specifies "1" for the drawing page and Line 140 draws a box.
Lines 150 through 200 switches the display page and alternately display the boxes drawn by Line 120 and 140.

SKIPF command

FUNCTION — Skips reading a file but checks the file.

FORMAT — SKIPF ["[CASO:] [<file name>]"]
(The abbreviated form is SK.)

STATEMENT EXAMPLE — SKIPF "PROG2"

EXPLANATION

This statement skips reading the specified file up to its end. When the specification of <file name> is omitted, reading of the first file on the tape is skipped.
Although this statement skips reading the file, it checks the file's contents. In other words, SKIPF has the same function as that of LOAD?

REFERENCE

FILES

SAMPLE PROGRAM

```
SAVE"SKIPF1" ················(1)

Ready
SAVE"SKIPF2" ················(2)

Ready
LIST

100 'SKIPF statement
110 MOTOR ON
140 PRINT" ﾃｰﾌﾟ ｦ ﾏｷﾓﾄﾞｼﾃｸﾀﾞｻｲ !!! "
150 STOP
160 SKIPF"CASO:SKIPF1"
170 MOTOR OFF
180 END

Ready
RUN
 ﾃｰﾌﾟ ｦ ﾏｷﾓﾄﾞｼﾃｸﾀﾞｻｲ !!!

Break In 150
Ready
CONT

Ready
```

The cassette tape is positioned at the start of the program saved by (2) among the programs saved on the cassette tape by (1) and (2). When a print out message is output by Line 140, the cassette tape is rewound.

SOUND

FUNCTION	Directly controls the sound generator and outputs sound.
FORMAT	SOUND <SG register No.>, <data> [, <SG No.>]
STATEMENT EXAMPLE	SOUND 13, 15

EXPLANATION

This statement sets data to SG (Sound Generator) and directly controls it by rewriting the contents.

A personal computer can have the function of a simple synthesizer by combining this instruction in various ways.

Specify an integer value in the range 0 ~ 13 for the <SG register No.>.

Specify the <data> with data in the range 0 ~ 255 for the interval, volume and noise.

With the specification of <SG No.>, select one of the two sound generators. Specify either "0" or "1". When this specification is omitted, "0" is selected. An option card is needed when specifying "1".

An SG consists of a tone generator + attenuator, noise generator, envelope generator and mixer. Use it by the following procedure.

(1) **Tone output** (Items 1 through 3 may be specified in any sequence)

1 Set the volume to "0". (The setting is not necessary when initializing.)
2 Set an interval.
3 Select a tone-noise channel.
4 Set a volume.

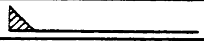
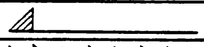
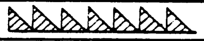
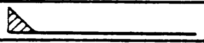
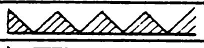
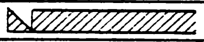
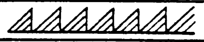
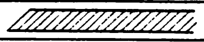
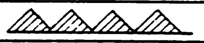
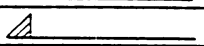
(2) **Envelope output** (Items 1 through 5 may be specified in any sequence.)

1 Set an interval.
2 Set a noise frequency.
3 Set a volume (&H10 ~ &H1F).
4 Select a tone-noise channel.
5 Set an envelope frequency.
6 Select an envelope waveform.

(Note) Since the sound is emitted when a volume is set for the tone and when an envelope waveform is selected for the envelope, if the above steps are conducted differently, unnecessary noise may be emitted.

Register No.	Data								Operation	Remarks
	7	6	5	4	3	2	1	0		
0	Lower 8-bit data								Channel A Scale	The scale is expressed by 12-bit data.
1					Higher 4-bit data					
2	Lower 8-bit data								Channel B Scale	
3					Higher 4-bit data					
4	Lower 8-bit data								Channel C Scale	
5					Higher 4-bit data					
6	5-bit data								Noise frequency	
7			Noise C	B	A	Tone C	B	A	Selection of noise tone channel	
8				M	4-bit data				Channel A Volume	When M=0, lower bits adjust the volume. When M=1, the envelope is effective.
9				M	4-bit data				Channel B Volume	
10				M	4-bit data				Channel C Volume	
11	Lower 8-bit data								Envelope period	
12	Higher 8-bit data									
13					As in the table below.				Envelope waveform	

- Register No. in the range 0 ~ 5 determines the oscillation frequency of the three oscillators. Use 0 and 1, 2 and 3, and 4 and 5 as pairs.
  12-bit data consisting of the upper 4 bits and lower 8 bits is effective.
  f = fclock/(16 x D), provided fclock = 1.008 MHz
  D stands for data to be written.
- Register No. 6 determines the average frequency of noise. Lower 5 bits are effective.
  f = fclock/(16 x D), provided fclock = 1.008 MHz
- Register No. 7 selects the channel. The selection is set when each corresponding bit is cleared ("0").
- Registers No. 8 ~ 10 set the volume of each channel. The lower 4 bits are effective and the maximum volume is 15. When Bit 4 is set to "1", the volume specification is made invalid by the envelope mode.
- Registers No. 11 ~ 13 control the envelope. The envelope waveform is set as shown in the table below.

	Lower 4 bits of Reg. No. 13				Envelope pattern (Vertical axis for volume, horizontal axis for time)
0 ~3	0	0	—	—	
4-7	0	1	—	—	
8	1	0	0	0	
9	1	0	0	1	
10	1	0	1	0	
11	1	0	1	1	
12	1	1	0	0	
13	1	1	0	1	
14	1	1	1	0	
15	1	1	1	1	

■ Period of envelope

$$f = \frac{fclock}{256 \times D}$$ provided D is data to be written to Registers No. 11 and 12.

$$T = \frac{1}{f}$$

NOTE

(1) When any one of the following is performed during execution of a SOUND statement, sound emission stops.

- When $\boxed{\text{CTRL}} + \boxed{\text{C}}$ or BREAK are pressed at a time other than when waiting for a BASIC command input.
- When $\boxed{\text{CTRL}} + \boxed{\text{D}}$ are pressed.
- When a RUN command is executed.
- When a NEW command is executed.
- When a LOAD command is executed.

REFERENCE

PLAY

SAMPLE PROGRAM

```
LIST

100 'SOUND statement
110 SOUND 7,&HFC
120 SOUND 8,12
130 SOUND 9,12
140 FOR I=1 TO 250
150 SOUND 0,I
160 SOUND 2,I
170 SOUND 1,0
180 SOUND 3,1
190 NEXT I
200 FOR I=1 TO 100
210 X=INT(RND(1)*256)
220 SOUND 0,X:SOUND 2,X
230 X=INT(RND(1)*2)+13:Y=X-INT(RND(1)*5)
240 SOUND 1,X:SOUND 3,Y
250 NEXT
260 SOUND 7,&HFF
270 END

Ready
RUN

Ready
```

This program directly controls the sound generator and outputs sound.

STOP

FUNCTION Temporarily stops program execution.

FORMAT STOP

STATEMENT EXAMPLE STOP

EXPLANATION

This statement temporarily stops program execution and the computer waits for a command. A STOP statement can be used at any place in a program to stop execution. When a STOP statement is executed, the following message is output.
  Break In <line No.>
The <line No.> here shows the line where the STOP statement was given. Program execution can be restarted by a CONT command.
A STOP statement is different from an END statement and does not close files.

REFERENCE

CONT

SAMPLE PROGRAM

```
LIST

100 'STOP statement
110 PRINT"Stopped. To continue hit 'cont'."
120 STOP
130 PRINT"Continued"
140 END

Ready
RUN
Stopped. To continue hit 'cont'.

Break In 120
Ready
CONT
Continued

Ready
```

Line 120 executes a STOP statement. After the execution is stopped, if CONT is input, execution restarts at Line 130.

STRIG(n) ON/OFF/STOP (Joystick trigger on/off/stop)

FUNCTION	Permits, inhibits or stops an interrupt from a joystick.
FORMAT	STRIG (<joystick No.>) ON OFF STOP
STATEMENT EXAMPLE	STRIG (1) ON

EXPLANATION

Specify a joystick with the <joystick No.>. "1" is for the button of Joystick A and "2" for the button of Joystick B.

(1) STRIG(n) ON   This permits an interrupt from the joystick. After this, each time the joystick is operated, an interrupt is generated and the execution branches to the interrupt routine defined by an ON STRIG(n) GOSUB statement.

(2) STRIG(n) OFF   This inhibits interrupts from the joystick. After this, the execution is not branched to an interrupt processing routine even if the joystick is operated.

(3) STRIG(n) STOP   This stops interrupts from the joystick. After this, the computer memorizes if the joystick is operated but execution is not branched to the interrupt processing routine. However, if a STRIG(n) ON statement is executed later, execution branches to the interrupt routine because of the joystick operation made earlier. A STRIG(n) STOP statement is effective only when the computer is in the STRIG(n) ON state.

NOTE

The STRIG(n) OFF state is set by executing a RUN, NEW, or CLEAR statement.

REFERENCE

ON STRIG(n) GOSUB, STICK function, and STRIG function

SAMPLE PROGRAM

```
LIST

100 'STRIG(N) ON/OFF/STOP
110 CLS :WIDTH 80
120 ON STRIG(1) GOSUB 250
130 STRIG(1) ON :PRINT "STRIG(1) ON を じっこう"
140 FOR I=1 TO 480
150   IF CSRLIN >6 THEN 190
160   PRINT "*";
170   FOR K=1 TO 10 :NEXT
180 NEXT
190 STRIG(1) OFF :PRINT "STRIG(1) OFF を じっこう"
200 FOR I=1 TO 480
210   PRINT "-";
220   FOR K=1 TO 10 :NEXT
230 NEXT
240 END
250 '------ Interrupt routine -------
260 PRINT
270 RETURN

Ready
```

```
STRIG(1) ON を じっこう
*********************************
*******************************
******************************************************
**************************
********************************************************************************
********************************************************************************
STRIG(1) OFF を じっこう
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

Ready
```

Line 120 defines an interrupt processing routine, Line 130 sets the STRIG (1) ON state, and "*" is displayed. When an interrupt is applied by pressing trigger button A, the line is fed at that point.
STRIG (1) OFF is executed after displaying "*" for six lines. After this, no line feed is conducted even if Trigger button A is pressed.

SWAP

FUNCTION Exchanges values of two variables.

FORMAT SWAP <variable name>, <variable name>

STATEMENT EXAMPLE SWAP A, B

EXPLANATION

This statement exchanges the values of two variables. The variables may have any form, but the two variables must have the form same.

SAMPLE PROGRAM

```
LIST

100 'SWAP statement
110 A=1:B=2:C=3
120 PRINT A;B;C
130 SWAP A,C
140 PRINT A;B;C
150 END

Ready
RUN
 1  2  3
 3  2  1

Ready
```

Line 130 exchanges the values of Variable A and Variable C and Line 140 displays them.

SYMBOL

FUNCTION	Displays a character string with an arbitrary size at an arbitrary position on a graphic screen.
FORMAT	SYMBOL (<Sx, Sy>), <character string> [.[<horizontal magnification>] [.[<vertical magnification>] [,[<color>] [,<plot option>]]]]
STATEMENT EXAMPLE	SYMBOL (100, 50), "A", 1, 1

EXPLANATION

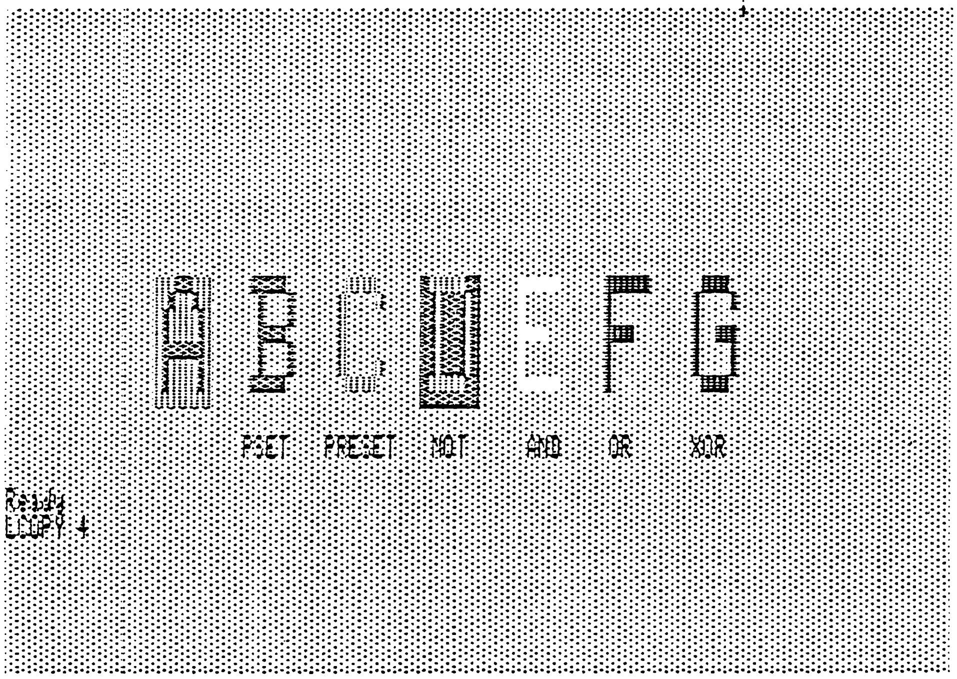
The <Sx, Sy> specifies the upper left dot at which the <character string> display starts using the screen coordinates.
The <character string> must be displayed inside the view port without missing any part.
The <horizontal magnification> and <vertical magnification> mean the horizontal and vertical magnifications of characters displayed and each character is displayed on the screen in a size of magnification x 8 dots. In the case of interlace mode, the vertical magnification must be an even number(Note). Specify an integer for the magnification. When it is omitted, "1" is assumed as the magnification.
If a negative value is specified for the <horizontal magnification>, a character that is inverted in the left-right direction is displayed. If a negative value is specified for the <vertical magnification>, a character inverted in the up-down direction is displayed. If a negative value is specified for both of the horizontal and vertical magnifications, a character is displayed inverted in the left-right and up-down directions. The magnification value is expressed as an absolute value.
The <color> shows the character color. If this specification is omitted, the character color of the COLOR statement that was executed immediately before is taken.
PSET, PRESET, AND, OR, XOR or NOT can be specified for the <plot option>. For details, refer to Format 2 of PUT@.
Note) When an odd number is specified as the vertical magnification, characters of non-interlace mode are output.

REFERENCE

POINT

SAMPLE PROGRAM

```
LIST

100 'SYMBOL statement
110 COLOR 7,4 :SCREEN 1,,1 :CLS
120 PAINT(0,0),1
130 SYMBOL(100,80),"A",5,5,2
140 SYMBOL(160,80),"B",5,5,2,PSET
150 SYMBOL(220,80),"C",5,5,2,PRESET
160 SYMBOL(280,80),"D",5,5,2,NOT
170 SYMBOL(340,80),"E",5,5,2,AND
180 SYMBOL(400,80),"F",5,5,2,OR
190 SYMBOL(460,80),"G",5,5,2,XOR
200 COLOR ,,1
210 LOCATE 20,16
215 PRINT "PSET   PRESET   NOT     AND     OR      XOR"
220 COLOR 7,0,0
230 END

Ready
```



```
                PSET  PRESET  NOT   AND   OR   XOR
Ready
LCOPY
```

Lines 130 through 190 display characters A through F in various ways by changing the plot option specification.

TERM command

FUNCTION Sets the system to the terminal mode.

FORMAT TERM ["<clock> [<word length> [<parity> [<stock bit length> [<mode> [<receive DEL code processing> [<RET key processing> [<receive CR code processing>]]]]]]]"]

STATEMENT EXAMPLE TERM "F7E1H"

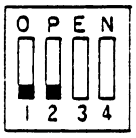
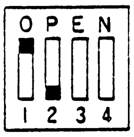
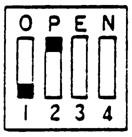
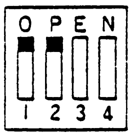
EXPLANATION

This statement changes the mode of the computer from the BASIC mode to the terminal mode. In the terminal mode, the computer can communicate with other computers or devices through the RS-232C interface. Specify one of the characters shown below for each item.

Item No.	Item	Specification char. of TERM command	Contents	Default
1	<clock>	F	Fact (1/16 of clock)	S
		S	Slow (1/64 of clock)	
2	<word length>	7	7 bits/char.	8 (Note)
		8	8 bits/char.	
3	<parity>	E	Even parity	N
		O	Odd parity	
		N	None parity	
4	<stop bit length>	1	1 bit	2
		2	2 bits	
5	<mode>	F	Full-duplex	F (Note)
		H	Half-duplex	
6	<receive DEL code processing>	B	Processed as BS code	U
		U	NUL code	
7	<RET key processing>	C	CR code transmission	C
		L	CR + LF sending	
8	<receive CR code processing>	R	Return operation + line feed operation	T
		T	Return operation	

(1) Selection of baud rate

The baud rate in the terminal mode can be selected by setting the TERM command clock and built-in DIP switch.

DIP switch	Specification of TERM command <clock>		Remarks
	S	F	
	300 baud	1200 baud	Set state when shipped from the factory
	600 baud	2400 baud	—
	1200 baud	4800 baud	—
	2400 baud	—	—

Note) DIP switches 3 and 4 are not related to baud rate setting.

(2) Limit on parity

The selectable <parity> is limited by the specifications of <word length> and <stop bit length>. The relationship between these is shown below. The ○ mark shows that the specification is possible and the × mark shows it is not possible.

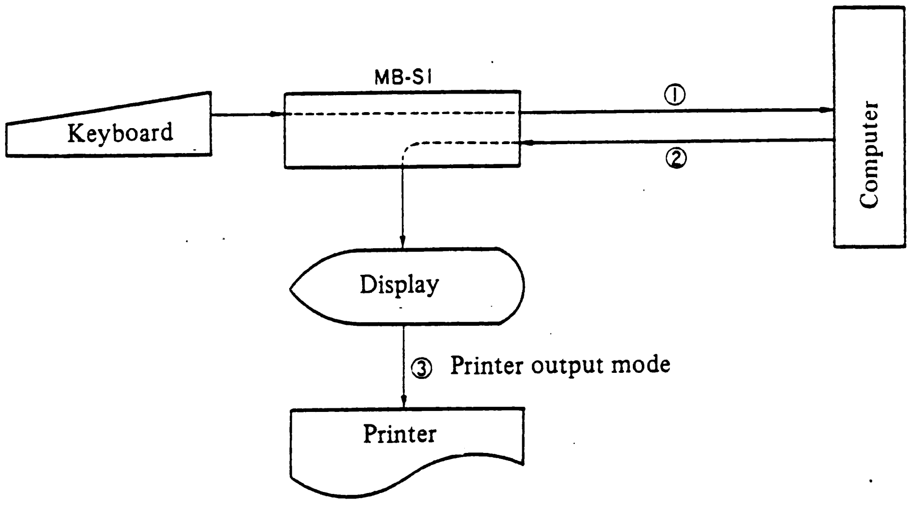
<word length>	<stop bit length>	<parity>		
		N: non-parity	O: Odd parity	E: Even parity
7	1	×	○	○
	2	×	○	○
8	1	○	○	○
	2	○	×	×

(3) Adding of shift codes

When 7 bits/characters is selected, S_I (shift in) and S_O (shift out) codes are automatically added at the start and end of character strings. Therefore, Katakana characters can be used (not Hiragana characters).

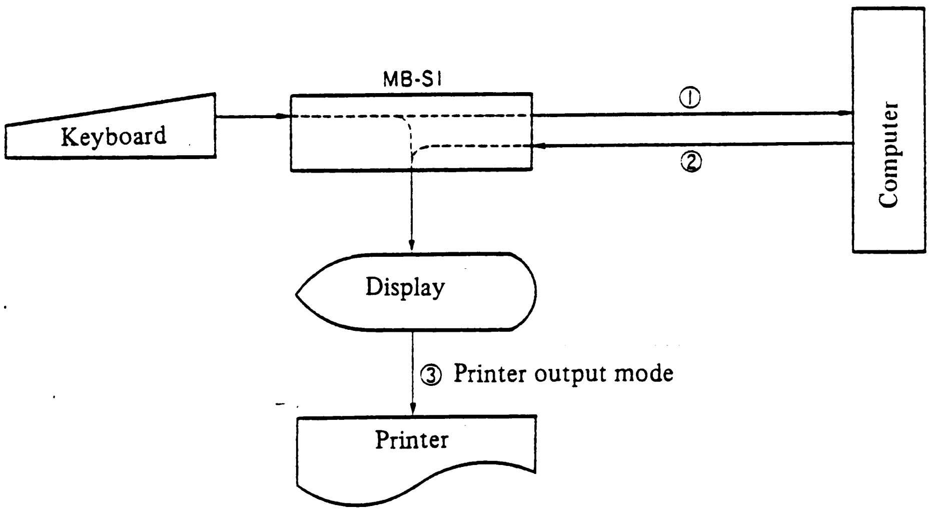
(4) Selection of full-duplex mode and half-duplex mode

- Full-duplex mode (when parameter "F" is selected)



1 Character codes input from the keyboard are sent to the line (not output to the display or printer).
2 The codes received from the line are shown on the display.
3 If the mode at this time is the printer output mode (pressing of COPY key for an odd number of times), the codes are also output to the printer.

- Half-duplex mode (when parameter "H" is selected)



1 Character codes input from the keyboard are output to the display screen and sent to the line.

2 The codes received from the line are shown on the display.

3 If the mode is printer output mode, the codes shown on the screen are also output to the printer.

(5) Receive DEL code processing

Received DEL code $(7F)_{16}$ is converted to BS code $(08)_{16}$, when B was specified, or to a null code when U was specified before it is processed.

(6) RET key processing

When the RETURN key (or CTRL + M) is pressed, if the specification given is C, OR code $(0D)_{16}$ is transmitted and if it is L, CR code and LF code $(0A)_{16}$ is transmitted.

(7) Receive CR code processing

When CR code $(0D)_{16}$ is received, return and linefeed operations are performed in the case of R. If the specification is T, only return operation is performed.

(8) Methods of using keys

The functions of keys in the conversation mode are shown below.

Item No.	Key	Function
1	$\boxed{\text{BREAK}}$ or $\boxed{\text{CTRL}}+\boxed{\text{C}}$	Sends a break signal (Note 1).
2	$\boxed{\text{COPY}}$ (Note 2)	Sets printer output mode. Pressing this key a second time resets the printer mode. This function can be used only when the baud rate is 300 baud.
3	$\boxed{\hookleftarrow}$	Returns the cursor and sends a CR code to the line.
4	$\boxed{\text{IS / BS}}$	Moves the cursor backward by one character and sends a BS code to the line. Outputs "\" (&H7F) in the printer output modes.
5	$\boxed{\text{CTRL}}$	When a character (symbol) key is pressed while this key is pressed, the corresponding control code is transmitted.
6	Alphanumerics	Sends out the corresponding code.
7	Katakana	Sends out the corresponding code.
8	Hiragana	Ignored.
9	Graphics	Sends out the corresponding code.

(Note 1) Signal of continuous space polarity of 250 ± 50 ms.

(Note 2) Since passwords are printed as they are, we recommend pressing the $\boxed{\text{COPY}}$ key after inputting a password. Also, the printer output is not done until receiving an LF code or the printer buffer is filled.

(9) Control codes

The control codes that are effective at the time of key input in the terminal mode are shown below. Press the CTRL + D keys to return to the BASIC mode.

Control code		Function	Input from keyboard
Hexa-decimal	Decimal		
07	7	Buzzer sounds	CTRL + G
08	8	Deletion (backspace)	CTRL + H
09	9	Moves to the next horizontal tab position	CTRL + I
0A	10	Line feed	CTRL + J
0B	11	Moves the cursor to the home position	CTRL + K
0C	12	Erases the display and moves the cursor to the home position	CTRL + L
0D	13	Carriage return	CTRL + M
14	20	Sets horizontal tab	CTRL + T
16	21	Displays the cursor	CTRL + V
17	23	Erases the cursor	CTRL + W
19	25	Resets horizontal tab	CTRL + Y
1C	28	Moves the cursor to the right	→
1D	29	Moves the cursor to the left	←
1E	30	Moves the cursor upward	↑
1F	31	Moves the cursor downward	↓

(10) Processing of received codes

Item No.	Receive code	S1 processing
1	$(00)_{16}$ ~ $(06)_{16}$	Ignored
2	BEL $(07)_{16}$	Bell sound
3	BS $(08)_{16}$	Deletion (backspace)
4	HT $(09)_{16}$	Moves to the next hor. tab position
5	LF $(0A)_{16}$	Line feed
6	HM $(0B)_{16}$	Ignored
7	CL $(0C)_{16}$	Ignored
8	CR $(0D)_{16}$	Carriage return
9	$(0E)_{16}$ ~ $(13)_{16}$	Ignored
10	$(14)_{16}$	Ignored
11	NK $(15)_{16}$	Ignored
12	SN $(16)_{16}$	Ignored
13	EB $(17)_{16}$	Ignored
14	CN $(18)_{16}$	Ignored
15	EM $(19)_{16}$	Ignored
16	$(1A)_{16}$ ~ $(1B)_{16}$	Ignored
17	→ $(1C)_{16}$	Ignored
18	← $(1D)_{16}$	Ignored
19	↑ $(1D)_{16}$	Ignored
20	↓ $(1F)_{16}$	Ignored
21	〒 $(A0)_{16}$	Converted to a space.
22	\ $(7F)_{16}$	* "B", backspace, "U", ignored
23	$(20)_{16}$ ~ $(FE)_{16}$	Displayed on screen as is
24	$(FF)_{16}$	Ignored

* When the parameter is "B", this code is converted to a backspace $(08)_{16}$.
When the parameter is "U", this code is ignored.

(11) When a line error occurs in reception, ■ is shown on the screen (output to the printer if in the printer output mode), and reception is continued.
Occurrence of the following states causes a line error:
Framing error:
State that the start bit and stop bit of data are not normal

Receiver overrun:

State in which the receive baud rate is lower than the baud rate at the other side.

Parity error:

When the sum of bits that are set to "1" in the received data does not match the even or odd parity whichever is set in advance.

Buffer overflow:

When received data comes in the state that the buffer (128 bytes) has no empty space.

(12) When 「〒」 esits at the time of reception, a blank is displayed.

NOTE

(1) When a TERM command is executed, the system is set to SCREEN 0, 0, 1. Therefore, Hiragana characters cannot be sent/received.

(2) Do not remove the receiver during communications using an acoustic coupler.

REFERENCE

ON COM(n) GOSUB, COM(n) ON/OFF/STOP

TRON/TROFF (Trace on/Trace off)

FUNCTION	Sets the trace mode.
FORMAT	TRON/TROFF
STATEMENT EXAMPLE	TRON

EXPLANATION

This statement is used for program debugging.
When TRON is executed, BASIC is set to the trace mode, and during program execution, the Line No. of an executed line is displayed on the screen enclosed in brackets ⌜[ ]⌟ each time execution moves to the next line. When any TROFF, NEW, NEW ON or LOAD is executed, the trace mode is reset.
By using this instruction and TROFF, the trace mode can be set and reset repeatedly without temporarily stopping program execution.

NOTE

Since the Line No. is displayed enclosed in brackets ⌜[ ]⌟ in the trace mode, the screen display is different from that made when the trace mode is reset.

SAMPLE PROGRAM

```
LIST

100 ' TRON, TROFF Statement
110 TRON
120 FOR I= 1 TO 10
130       PRINT I
140 IF I < 5 THEN 150 ELSE TROFF
150 NEXT I
160 END

Ready
RUN
[120][130] 1
[140][150][130] 2
[140][150][130] 3
[140][150][130] 4
[140][150][130] 5
[140] 6
 7
 8
 9
 10

Ready
```

The parts enclosed in brackets ⌜[ ]⌟ are the line numbers executed.

UNLIST command

FUNCTION	Prohibits list output.
FORMAT	UNLIST [<line No.>]
STATEMENT EXAMPLE	UNLIST 1000

EXPLANATION

When an UNLIST statement with a specified <line No.>, output of the list after the specified <line No.> is inhibited. When the <line No.> specification is omitted, the output of the whole program list is inhibited. Any line that is input after this command is included in the list output to be inhibited if the line No. of such a line is greater than the specified <line No.>.
Note that this state is not reset until the power switch is turned off or a NEW, NEW ON or LOAD command is executed.
Also note that with a program in which UNLIST was executed and which was once saved and then loaded, the effect of inhibiting the list output is active.

NOTE

After executing an UNLIST command, if a new program is to be input (from the keyboard), make sure to execute a NEW or NEW ON command prior to input.
Also, note that, if the spelling ⌜UNLIST⌟ exists at the start of a statement (a variable name at the left side of a LET statement in which ⌜LET⌟ is omitted), even if an error occurs, UNLIST may be executed.

REFERENCE

LIST

SAMPLE PROGRAM

```
Ready
LIST

100 'UNLIST statement
110 UNLIST 500
120 GOSUB 500
130 FOR I=1 TO 10
140 PRINT I;
150 NEXT I
160 END
500 CLS
510 LOCATE 10,10
520 PRINT "HITACHI COUNT ROUTINE"
530 RETURN

Ready


                HITACHI COUNT ROUTINE
 1  2  3  4  5  6  7  8  9  10
Ready



LIST

100 'UNLIST statement
110 UNLIST 500
120 GOSUB 500
130 FOR I=1 TO 10
140 PRINT I;
150 NEXT I
160 END

Ready
```

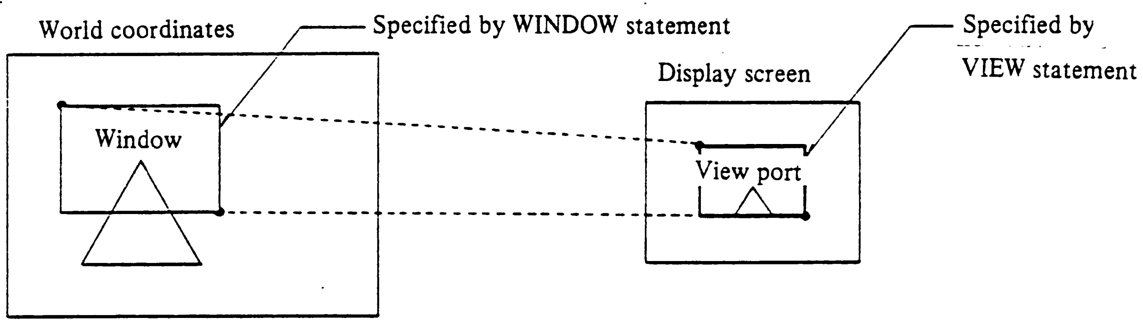
After executing the above program, if ⌜LIST⌟ is typed in, the line contents after Line 500 are not displayed.

VIEW

FUNCTION	Specifies a view port on the display screen.
FORMAT	VIEW (<Dx$_1$, Dy$_1$>) – (<Dx$_2$, Dy$_2$>) [, [<area color>] [, <boundary color>]]
STATEMENT EXAMPLE	VIEW (100, 50) – (200, 150), 5, 6

EXPLANATION

A VIEW statement specifies the drawing display area (view port) on the display screen. Specify a drawing display area in a rectangle having <Dx$_1$, Dy$_1$> of the display coordinate system as the upper left point and <Dx$_2$, Dy$_2$> as the lower right point. A view port must be specified within the range that is specified by the fineness mode of a SCREEN statement. In other words, in the case of SCREEN 1, the view port is in the range (0, 0) – (639, 199) and in the case of SCREEN 0, it is within (0, 0) – (319, 199).
A drawing within a world coordinate system specified by a WINDOW statement is displayed in the view port specified by a VIEW statement.



When the <area color> is specified, the inside of the view port is colored by the specified Palette No.
When the <boundary color> is specified, the frame of view port is colored by the specified Palette No. If the inside of the frame is cleared by a CLS3 statement, only the frame remains. The frame is drawn one dot outside the view port.
When set, a view port does not change until a VIEW or SCREEN statement is input. Also, it sets LP (refer to 1.10 Coordinate system) at the upper left point of the view port.
In the initial state, a view port is set to the whole display screen.

NOTE

Since a VIEW statement only specifies a drawing display area on the display screen, it does not move the drawing of any preceeding view port. Also note that no drawing is displayed outside a view port.

REFERENCE

WINDOW

WHILE ~ WEND (While ~ Wend)

FUNCTION	Executes repeatedly while the given conditions are satisfied.
FORMAT	WHILE <qualification> ∫ WEND
STATEMENT EXAMPLE	WHILE A > 100 ∫ WEND

EXPLANATION

During the time the <qualification> is true (not "0"), statements between the WHILE statement and WEND statement are executed, and after executing the WEND statement, execution returns to the WHILE statement. The value of <qualification> is checked again, and if it is true, the same operation is conducted. If the <qualification> is not true (that is, it is "0"), execution is transferred to the statement after the WEND statement.
If the result of checking the value of <qualification> the first time is not true ("0"), the statements between the WHILE and WEND statements are never executed.
A loop of WHILE ~ WEND statements can be structured with as many nests as required, which is the same as a loop of FOR ~ NEXT statements. Each WEND statement corresponds to a WHILE statement that is written before it and closest to it, which is also the same as the nested structure of FOR ~ NEXT statements.
A WHILE statement and WEND statement must always be a pair.

REFERENCE

IF ~ THEN ~ ELSE, FOR ~ NEXT

SAMPLE PROGRAM

```
Ready
LIST

100 'WHILE WEND statement
110 I=1
120 WHILE I< 6
130      J=1
140     WHILE J< 4
150        PRINT SPC(I);"HITACHI";
160        J=J+1
170    WEND
180    PRINT "  Personal Computer"
190    I=I+1
200 WEND

Ready
RUN
 HITACHI HITACHI HITACHI  Personal Computer
  HITACHI  HITACHI  HITACHI  Personal Computer
   HITACHI   HITACHI   HITACHI  Personal Computer
    HITACHI    HITACHI    HITACHI  Personal Computer
     HITACHI     HITACHI     HITACHI  Personal Computer

Ready
```

During the time the conditions of the WHILE ~ WEND loop on Lines from 120 to 200 are satisfied, characters are displayed by lines 150 and 180.

WIDTH

FUNCTION Specifies a number of characters to be displayed per line.

FORMAT

1 WIDTH $\left|\begin{array}{c}40\\80\end{array}\right|$

2 WIDTH <device name>, <output width>

STATEMENT EXAMPLE

```
1  WIDTH 80
2  WIDTH "LPT0:", 80
```

EXPLANATION

1 This sets the number of characters per line of the screen. 40 characters/line or 80 characters/line can be selected. When a WIDTH statement is executed, the display on the screen disappears temporarily.

2 This sets the output width of the device specified by the <device name>. Specify any one of 「SCRN:」, 「LPT0:」 ~ 「LPT2:」, or 「COM0:」 ~ 「COM4:」 for the <device name>. If 「SCRN:」 is specified for the <device name>, the result is the same as that of Format 1 . Specify the <output width> with a value in the range 1 ~ 255.

(1) PRINT/LPRINT/PRINT # statement

When using a semi-colon 「;」 as a delimiter: If numeric values or character strings to be output exceed the specified output width, line feed is performed first and these numeric values or character strings are output from the start of a new line.

When using a comma 「,」 as a delimiter: The output is made in groups of 14 digits. If the output width cannot be devided by 14 digits, the remaining area less than 14 digits at the end of the line is incorporated into the last 14-digit area. If, however, outputting numeric values or character strings starting at the next area of the current line causes the output to exceed the output width, instead of using the area of the current line, line feed is performed first and the remaining data is output from the start of the next line.

(2) PRINT USING/LPRINT USING/PRINT # USING statement

The control described in (1) is not performed. Instead, as the output width is reached, the line is fed mechanically. Therefore, in this case, one numeric value or character string may be output split into two lines.

(3) Statements other than the above (LIST, LLIST, etc.)

When the output width is reached, the line is fed mechanically.

If "255" is specified as the output width, the prearranged line feed actions as described in (1) through (3) are suppressed, and no line feed is conducted unless the program specifies it. However, in the case of screen and printer, the hardware performs line feed as the number of characters reached the physical output limit.

3 The initial value of <output width> are as follows:

「SCRN : 」	40
「LPT0 : 」~「LPT2 : 」	80
「COM0 : 」~「COM4 : 」	255

REFERENCE

SCREEN

SAMPLE PROGRAM

```
Ready
LIST

100 'WIDTH statement
110 A$="123456789012345"
130 WIDTH "LPT0:",5
140 LPRINT A$
142 WIDTH "LPT0:",3
144 LPRINT A$
160 END

Ready


12345
67890
12345
123
456
789
012
345
```

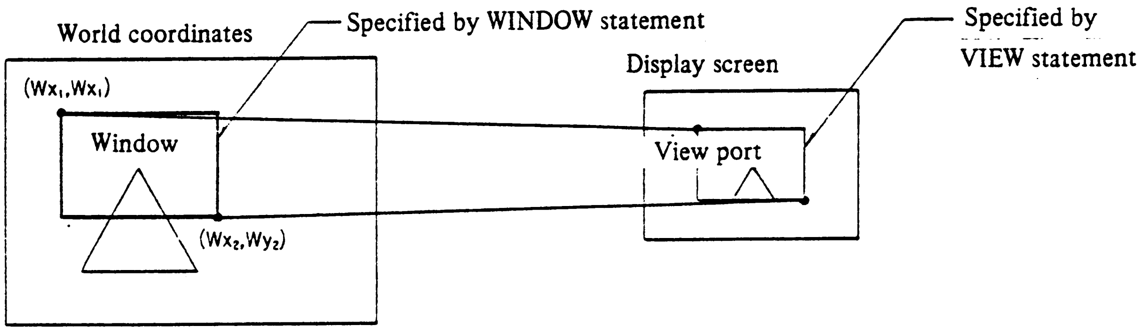
Lines 130 and 142 change the output width and the A$ contents are output to a printer.

WINDOW

FUNCTION	Specifies an area in world coordinates to be displayed in a view port.
FORMAT	WINDOW (<Wx_1, Wy_1>) – (<Wx_2, Wy_2>)
STATEMENT EXAMPLE	WINDOW (–500, –600) – (100, 200)

EXPLANATION

This statement allocates a rectangular area having <Wx_1, Wy_1> and <Wx_2, Wy_2> as its diagonal points. This rectangle is called a window. The relation here is that the outside scene (world coordinate system) is looked at through a window (rectangle). The visible area changes as the position and size of the window are changed.



Specify the window coordinates using world coordinates. Once set, they remain unchanged until a new WINDOW or SCREEN statement is given. Also, LP (refer to 1.10 Coordinate system) is set at the upper left point.
In the initial state, the window is set to (0, 0) – (639, 199).

NOTE

A WINDOW statement specifies an area in world coordinates. Changing the window position does not cause any change on the drawing displayed in a view port.

REFERENCE

VIEW

SAMPLE PROGRAM

```
LIST

100 'WINDOW statement
110 CLS:SCREEN 1:WIDTH 80
120 VIEW(1,1)-(638,198),,2
130 LINE(0,0)-(639,199),PSET,6
150 WINDOW(0,0)-(4733,1474)
160 VIEW(15,15)-(300,190),,2
170 GOSUB 500
180 LOCATE 5,5:PRINT"VIEW ﾎﾟｰﾄ !!!"
190 VIEW(30,30)-(150,150),,2
200 GOSUB 500
210 VIEW(80,70)-(90,80),,2
220 GOSUB 500
230 END
500 LINE(0,0)-(4733,1473),PSET,6
510 RETURN

Ready
```

This program draws a diagonal of the view port while changing the view port size.

A

ABS function (Absolute function)

FUNCTION	Gives an absolute value.
FORMAT	ABS (<argument>)
STATEMENT EXAMPLE	X = ABS (A + B)

EXPLANATION

This statement gives the absolute value of the <argument>.

ABS (X) = | X |

SAMPLE PROGRAM

```
Ready
LIST

100 'ABS function
110 PRINT USING "&     &":"    I","ABS(I)"
120 FOR I=-3 TO  3 STEP  .5
130 PRINT USING "###.#":I,ABS(I)
140 NEXT I
150 END

Ready


RUN
   I  ABS(I)
 -3.0  3.0
 -2.5  2.5
 -2.0  2.0
 -1.5  1.5
 -1.0  1.0
 -0.5  0.5
  0.0  0.0
  0.5  0.5
  1.0  1.0
  1.5  1.5
  2.0  2.0
  2.5  2.5
  3.0  3.0

Ready
```

This program displays the value and absolute value of Variable "I" while changing "I" from "–3" to "3" in steps of 0.5 by the FOR ~ NEXT statements starting at Line 120.

ASC function (ASCII function)

FUNCTION Gives a character code.

FORMAT ASC (<character expression>)

STATEMENT EXAMPLE X = ASC ("A")

EXPLANATION

This statement gives a character code to the starting character of <character expression>. Refer to the Character Code Table for the correspondence between characters and character codes.
Null strings cannot be specified as the <character expression>.

REFERENCE

CHR$ function

SAMPLE PROGRAM

```
Ready
LIST

100 'ASC function
110 FOR I=1 TO 18
120 READ C$,D$
130 PRINT C$;ASC(C$),D$;ASC(D$)
140 NEXT I
150 DATA A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,
    V,W,X,Y,Z
160 DATA 1,2,3,4,5,6,7,8,9,0
170 END

Ready


RUN
A 65              B 66
C 67              D 68
E 69              F 70
G 71              H 72
I 73              J 74
K 75              L 76
M 77              N 78
O 79              P 80
Q 81              R 82
S 83              T 84
U 85              V 86
W 87              X 88
Y 89              Z 90
1 49              2 50
3 51              4 52
5 53              6 54
7 55              8 56
9 57              0 48

Ready
```

The FOR ~ NEXT statements starting at Line 110 read the characters in the DATA statements on Lines 150 and 160 and display the characters and the character codes.

ATN function (Arc tangent function)

FUNCTION	Gives an arc tangent.
FORMAT	ATN (<argument>)
STATEMENT EXAMPLE	X = ATN (1/2)

EXPLANATION

This statement gives the arc tangent of the <argument> in radians. The range of numeric values that can be given is from $-\pi/2$ to $\pi/2$. ATN functions are operated in single precision.

ATN (X) $\equiv \tan^{-1}X$

REFERENCE

TAN function, COS function, SIN function

SAMPLE PROGRAM

```
Ready
LIST

100 'ATN function
110 PI=3.1415926#
120 PRINT"    X[RAD]     TAN(X)     ATN(TAN(X))"
130 FOR X=-PI/4 TO PI/4 STEP PI/4
140  TN=TAN(X)
150  PRINT USING"##.####### ########   ##.#######";
    X,TN,ATN(TN)
160 NEXT X

Ready
RUN
   X[RAD]     TAN(X)    ATN(TAN(X))
-0.7853980        -1   -0.7853980
 0.0000000         0    0.0000000
 0.7853980         1    0.7853980

Ready
```

. This programs shows the values of X, TAN (X), ATN (TAN (X)) while increasing the X value from $-\frac{\pi}{4}$ in steps of $\frac{\pi}{4}$.

CDBL function (Convert double function)

FUNCTION	Converts to the double precision type.
FORMAT	CDBL (<argument>)
STATEMENT EXAMPLE	A # = CDBL (B%)

EXPLANATION

This statement converts the <argument> to double-precision type.

REFERENCE

CINT function, CSNG function

SAMPLE PROGRAM

```
LIST

100 'CDBL function
110 A=1.1 : B=3.3
120 C#=A/B
130 D#=CDBL(A)/CDBL(B)
140 PRINT "A =";A,"B= ";B
150 PRINT "C#=";C#
160 PRINT "D#=";D#
170 END

Ready
RUN
A = 1.1         B=  3.3
C#= .3333333432674408
D#= .3333333453746759

Ready
```

This is an example of converting an argument to the double-precision type by the CDBL function.
The program shows the result of single precision division while setting "A" to "1.1" and "B" to "3.3" as well as the result of converting "A" and "B" to double-precision and division.

CHR$ function (Character dollar function)

FUNCTION — Gives the character that corresponds to a character code.

FORMAT — CHR$ (<argument>)

STATEMENT EXAMPLE — A$ = CHR$ (66)

EXPLANATION

This statement gives the character that corresponds to the specified <argument> in the range 0 ~ 255.
For information on character codes, refer to the Character Code Table. Generally, the CHR$ function is used to output special characters.

REFERENCE

ASC function

SAMPLE PROGRAM

```
Ready
LIST

100 'CHR$ function
110 FOR I = ASC(" ")  TO  ASC("Z")
120   PRINT I;CHR$(I),
130 NEXT I
140 END

Ready
RUN
 32          33 !        34 "        35 #        36 $
 37 %        38 &        39 '        40 (        41 )
 42 *        43 +        44 ,        45 -        46 .
 47 /        48 0        49 1        50 2        51 3
 52 4        53 5        54 6        55 7        56 8
 57 9        58 :        59 ;        60 <        61 =
 62 >        63 ?        64 @        65 A        66 B
 67 C        68 D        69 E        70 F        71 G
 72 H        73 I        74 J        75 K        76 L
 77 M        78 N        79 O        80 P        81 Q
 82 R        83 S        84 T        85 U        86 V
 87 W        88 X        89 Y        90 Z
Ready
```

This is an example of outputting characters that correspond with character codes by the CHR$ function.
The FOR ~ NEXT statements on lines 110 through 130 display the character codes and corresponding characters ranging from a space to all alphabetic characters A through Z.

CINT function (Convert integer function)

FUNCTION Converts to integer type.

FORMAT CINT (<argument>)

STATEMENT EXAMPLE A% = CINT (B #)

EXPLANATION

This statement converts the specified <argument> by counting .5 and over of fractions up to whole numbers and truncaties the rest. If the conversion result does not fall in a range of –32768 to 32767, an error occurs in.

REFERENCE

INT function, FIX function

SAMPLE PROGRAM

```
LIST

100 'CINT function
110 A! = 10/3
120 B# = -13.9/2
130 PRINT"A!          =";A!,"B#        = ";B#
140 PRINT"CINT(A!)   =";CINT(A!);SPC(14);
    "CINT(B#) = ";CINT(B#)
150 END

Ready
RUN
A!          = 3.33333       B#          = -6.949999809265137
CINT(A!)    = 3             CINT(B#)    = -7

Ready
```

This program converts single precision and double precision numerics to integers and displays them. The result of the CINT function's operation is the integer type of two bytes, and as shown by the execution result, the fractions are handled by counting .5 and over of fractions as whole numbers and truncating the rest.

COS function (Cosine function)

FUNCTION	Gives a cosine.
FORMAT	COS (<cosine>)
STATEMENT EXAMPLE	X = COS (N * PI)

EXPLANATION

This statement gives a cosine for the <argument> in radian units. The COS function operation is conducted in single precision.

COS (X) ≡ cosX

SAMPLE PROGRAM

```
Ready
LIST

100 'COS function
110 PI=3.14159
120 CLS
130 FOR I = 0 TO 720 STEP  30
140      T=I*PI/180
150      N=10*COS(T)+45
160      PRINT I;TAB(N);"."
170 NEXT I
180 END

Ready

0                                                      .
30                                                   .
60                                                .
90                                           .
120                                     .
150                                 .
180                                .
210                                 .
240                                     .
270                                          .
300                                               .
330                                                  .
360                                                    .
390                                                  .
420                                               .
450                                          .
480                                     .
510                                 .
540                                .
570                                 .
600                                     .
630                                          .
660                                               .
690                                                  .
720                                                    .

Ready
```

This program draws a cosine curve using the COS function.
The FOR ~ NEXT statements starting at Line 130 calculate the cosine values for two cycles for 0° to 720° at every 30°, outputting the equivalent number of spaces with the TAB function, and outputs 「.」.
Line 140 is an expression used to convert degrees to radians since the unit of the COS function argument is radians.

CSNG function (Convert single function)

FUNCTION	Converts a numeric value to a single-precision type numeric.
FORMAT	CSNG (<argument>)
STATEMENT EXAMPLE	AI = CSNG (B #)

EXPLANATION

This statement converts the specified <argument> to the single-precision type.

REFERENCE

CSNG function, CDBL function

SAMPLE PROGRAM

```
LIST

100 'CSNG function
110 A%=1:B#=2/3
120 PRINT"A%        = ";A%;SPC(5);"B#        = ";B#
130 PRINT"CSNG(A%) = ";CSNG(A%);SPC(5);
   "CSNG(B#) = ";CSNG(B#)
140 END

Ready
RUN
A%        =  1      B#        =  .6666666865348816
CSNG(A%) =  1      CSNG(B#) =  .666667

Ready
```

This program converts the values of integer type A% and double-precision type B# to single-precision type values and displays the results.
Although the execution results of the A% value and CSNG (A%) are the same on the display, in the internal expression A% is two bytes and CSNG (A%) is four bytes.

CSRLIN function (Cursor line function)

FUNCTION Gives the vertical position of the cursor on the screen.

FORMAT CSRLIN

STATEMENT EXAMPLE X = CSRLIN

EXPLANATION

This statement gives the current vertical position of cursor in the line unit of 0 ~ 24. No argument is used.

REFERENCE

POS function

SAMPLE PROGRAM

```
Ready
LIST

100 'CSRLIN system parameter
110 PRINT CSRLIN
120 PRINT CHR$(13)
130 PRINT CSRLIN
140 END

Ready
RUN
 12

 14

Ready
```

This is an example of a program that sets the cursor vertical position using the CSLIN function.
This program first erases the screen by CLS, LIST is instructed and the CSLIN function execution result is displayed.
After inputting a RUN command, the cursor is on the 12th line from the top line of the screen. Since Line 110 is executed in this state, "12" is displayed and the cursor moves to the next line. Since the carriage return code is output by Line 120, the cursor moves to the next line. At this time, the cursor is on the 14th line from the top line of the screen, and since Line 130 is executed at this point, the displayed is changed to "14".

CVI, CVS, or CVD function (Convert integer, single or double function)

FUNCTION	Converts character type numeric data to numeric type data.
FORMAT	CVI (<2-byte character string>) CVS (<4-byte character string>) CSD (<8-byte character string>)
STATEMENT EXAMPLE	1 A% = CVI (A$) 2 B = CVS (CHR$ (&H82) + STRING$ (3, 0)) 3 C# = CVD (CHR$ (&H80) + STRING$ (7,&H88))

EXPLANATION

Character type data converted from numeric type data by the MKI$, MIS$ or MKD$ function can be changed back to the initial numeric type by CVI, CVS or CVD function. The CVI function is applied to integers, CVS function to single-precision real numbers and the CVD function to double-precision real numbers.

REFERENCE

MKI$ function, MKS$ function, MKD$ function

D

DATE function

FUNCTION — Gives the number of days counted by the built-in clock.

FORMAT — DATE

STATEMENT EXAMPLE — A = DATE

EXPLANATION

This statement reads the total number of days counting January 1 of each year as "1". Since the value given by the DATE function is interlocked with the value of DATE$, set the DATE function value by assigning a value to DATE$.

```
2000  MMSS=VAL (MID$ (DATE$, 4, 2))
2010  MMDD=VAL (MID$ (DATE$, 7, 2))
2020  IF (LEFT$ (DATE$, 2) = "80" OR
LEFT$ (DATE$, 2)= "84" OR LEFT$ (DATE$, 2)= "88" ) AND (MMSS=2 AND
MMDD=29)
THEN DD=DATE-1  ELSE DD=DATE
```

REFERENCE

TIME function, DATE (system variable)

SAMPLE PROGRAM

```
Ready
LIST

100 'DATE system parameter
110 DATE$="83/01/17"
120 PRINT DATE$,DATE
130 END

Ready
RUN
83/01/17         17

Ready
```

This program displays the present date using the DATE$ function. The DATE$ variable is useful in a program that needs the date, for example, a stock control program.

DATES (System variable)

FUNCTION — Gives the date from the built-in clock.

FORMAT — DATE$

STATEMENT EXAMPLE — D$ = DATES
DATE$ = "83/01/17"

EXPLANATION

This statement reads the date from the built-in clock. The date is expressed as a character string in the following format.

YY/MM/DD

"YY" shows the lower two digits of the year AD. "MM" shows the mouth and "DD" the day. Assignment can be done in the same format. If "YY", "MM" or "DD" is a single digit at the time of setting, specify "0" as the ten's digit to make it to two digits. The effective period of calculation including leap years is 1980 to 2000.

REFERENCE

TIMES (System variable), DATE function

SAMPLE PROGRAM

```
Ready
LIST

100 'DATE$ system parameter
110 PRINT "Today is ";DATE$;"."
120 END

Ready
RUN
Today is 83/01/17.

Ready
```

EOF function (End of file function)

FUNCTION	Determines whether or not a file has terminated.
FORMAT	EOF (<file No.>)
STATEMENT EXAMPLE	X = EOF (1)

EXPLANATION

This statement detects the EOF mark written at the end of an input file specified by the <file No.>. When EOF is detected, truth (−1) is specified, and if not detected, non-truth (0) is specified.
If the specified file is an RS-232C port, this statement checks whether the buffer allocated as the channel is empty, and if it is empty, truth is specified. If not empty, non-truth is specified.

SAMPLE PROGRAM

```
Ready
LIST

100 'EOF function
110 OPEN"I",#1,"DATA"
120 DIM NUM(100)
130 COUNT=1
140 IF EOF(1) THEN 180
150 INPUT #1,NUM(COUNT)
160 COUNT=COUNT+1
170 GOTO  140
180 PRINT"END OF FILE,DATA IS ENTERED."
190 CLOSE #1
200 FOR J=1 TO  100
210       PRINT  NUM(J),
220 NEXT J
230 END

Ready

RUN
END OF FILE,DATA IS ENTERED.
 1              2              3              4              5
 6              7              8              9              10
 11             12             13             14             15
 16             17             18             19             20
 21             22             23             24             25
 26             27             28             29             30
 31             32             33             34             35
 36             37             38             39             40
 41             42             43             44             45
 46             47             48             49             50
 51             52             53             54             55
 56             57             58             59             60
 61             62             63             64             65
 66             67             68             69             70
 71             72             73             74             75
 76             77             78             79             80
 81             82             83             84             85
 86             87             88             89             90
 91             92             93             94             95
 96             97             98             99             100

Ready
```

"DATA" is a sequential file and it contains natural numbers in a range of 1 to 100. This program reads data from DATA and displays it.
Line 110 inputs the DATA file and opens it. The DIM statement on Line 120 is a declaration of an array to which the data is to be read. Line 130 initializes variable COUNT that is used as a subscript of the array. The EOF function is used for the logical expression of the IF statement on line 140. If the value of the EOF function is true, that is, when the file terminates, the execution is transferred to Line 180. Otherwise, the execution goes to Line 150, reads the data, adds one to the COUNT value on executing Line 160, and then returns to Line 140 as instructed by Line 170. The data is read into array NUM in this loop until the file comes to an end. When the file comes to an end, execution is transferred to Line 180, and reading of the file is notified. Line 190 closes the file and the FOR ~ NEXT statements of Lines 200 and 220 display the data that has been read into array NUM.

ERL function (Error line function)

FUNCTION Gives the Line No. where an error occurred.

FORMAT ERL

STATEMENT EXAMPLE Y = ERL

EXPLANATION

This statement gives the Line No. where an error has occurred. 65535 is given when an error occurred during the time the computer is waiting for a command.
This function is very useful to check where the error occurred using an error recovery processing routine.

NOTE

To check the value of an ERL function using an IF statement, write the Line No. on the right side of the equals sign. If it is written on the left side, the Line No. is not corrected when a RENUM command is executed.

REFERENCE

ON ERROR GOTO, ERR function, ERROR

SAMPLE PROGRAM

```
LIST

100 'ERL System parameter
110 '
120 ON ERROR GOTO 500
130 INPUT "X=",X
140 INPUT "Y=",Y
150 Z=X/Y:PRINT"X/Y=";Z
160 END
500 'ERROR ROUTINE
510 A$="Can't divide!      Input once more."
520 IF ERL=150 AND ERR=11 THEN PRINT A$:RESUME 140
530 PRINT "ERR=";ERR;" ERROR OCCURED IN ";ERL
540 END

Ready
RUN
X=2
Y=0
Can't divide!      Input once more.
Y=2
X/Y= 1

Ready
RUN
X=123
Y=123E99
ERR= 6  ERROR OCCURED IN  140

Ready
```

This program displays the Line No. where an error occurred.
Key input numeric values on Lines 130 and 140. If the result of division by Line 150 is an error, the Error No. and Line No. where the error occurred are displayed by Lines 500 through 530.

ERR function (Error function)

FUNCTION Gives an error code to a generated error.

FORMAT ERR

STATEMENT EXAMPLE IF ERR = 10 GOTO 100

EXPLANATION

This statement gives an error code to a generated error. This function is useful to check what kind of error occurred in an error recovery processing routine (specified by an ON ERROR GOTO statement).
For information of error codes and their meaning, refer to the Error Messages, provided in the APPENDIX.

REFERENCE

ERROR, ON ERROR GOTO, ERL function

SAMPLE PROGRAM

```
Ready
LIST

100 ' ERR function
110 ON ERROR GOTO 500
120 PRIN
130 LOCATE 100,400
140 END
500 '
510 PRINT "ｴﾗｰ ｺｰﾄﾞ ": ERR
520 RESUME NEXT

Ready
RUN
ｴﾗｰ ｺｰﾄﾞ  2
ｴﾗｰ ｺｰﾄﾞ  5

Ready
```

Line 110 specifies the Line No. of the error interrupt, and Line 510 displays the Error No.

EXP function (Exponential function)

FUNCTION	Gives the power number of "e"
FORMAT	EXP (<argument>)
STATEMENT EXAMPLE	X = EXP (10)

EXPLANATION

This statement gives the <argument> power of "e". The operation of EXP function is conducted in single precision.

$$\text{EXP (X)} \equiv e^{x}$$

SAMPLE PROGRAM

```
Ready
LIST

100 'EXP function
110 INPUT X
120 Y=EXP(X)
130 PRINT Y
140 END

Ready
RUN
? 10
 22026.5

Ready
```

This program obtains the X-power of "e".
The INPUT statement of Line 110 inputs the X value, the EXP function of Line 120 calculates the X-power of "e", and Line 130 outputs it. The EXP function is useful to make hyperbolic functions.

FIX function

FUNCTION Gives the integer part.

FORMAT FIX (<argument>)

STATEMENT EXAMPLE A = FIX (−10.5)

EXPLANATION

This gives the integer part of the <argument>.

REFERENCE

INT function, CINT function

SAMPLE PROGRAM

```
Ready
LIST

100 'FIX function
110 PRINT USING "&   &";"  X","FIX(X)","INT(X)"
120 FOR X=-2 TO 2 STEP .5
130 PRINT USING "##.# ###### ######";X,FIX(X),INT(X)
140 NEXT X
150 END

Ready
RUN
  X     FIX(X)  INT(X)
-2.0      -2      -2
-1.5      -1      -2
-1.0      -1      -1
-0.5       0      -1
 0.0       0       0
 0.5       0       0
 1.0       1       1
 1.5       1       1
 2.0       2       2

Ready
```

This program compares the FIX function and INT function so that the difference between the two can be seen. The FOR ~ NEXT statements starting at Line 120 increase the X value from "−2" to "2" in steps of 0.5, and the results of the FIX function and INT function are displayed.
While the FIX function returns the value of the argument with its fractions truncated, the INT function returns an integer that is the largest but not exceeding the argument. Therefore, if X is "−1.5", for example, FIX (X) makes it "−1" but INT (X) makes it "−2".

FN function (Function)

FUNCTION	Gives the value of a user function.
FORMAT	FN <name> [(<real argument> [, <real argument> ...])]
STATEMENT EXAMPLE	FNA (50, "ABC")

EXPLANATION

This statement gives the value of a user function that is defined by a DEF FN statement. A calculation expression must be defined by a DEF FN statement in advance.
This specification of <name> must satisfy the conditions that a variable name should satisfy (refer to 1.7 Variables).
The <read argument> can be specified by an expression (including character expression). The dummy argument and type (numeric or character) of the DEF FN statement must be the same. The operation result of replacing each dummy argument in the definition statement of the DEF FN statement with the corresponding real argument is given as the value of the function.

REFERENCE

DEF FN

SAMPLE PROGRAM

```
Ready
LIST

100 'FN function
110 DEF FNAREA(R)=SQR(2)*R
120 READ R
130 PRINT FNAREA(R)
132 DATA 2
140 END

Ready
RUN
 2.82843

Ready
```

Line 110 defines the user function. Line 130 calculates the function and outputs the result based on the real argument input by Line 120.

FRE function (Free function)

FUNCTION — Gives the unused area of a variable, array, test or character.

FORMAT — FRE (<argument>)

STATEMENT EXAMPLE — A = FRE (1)

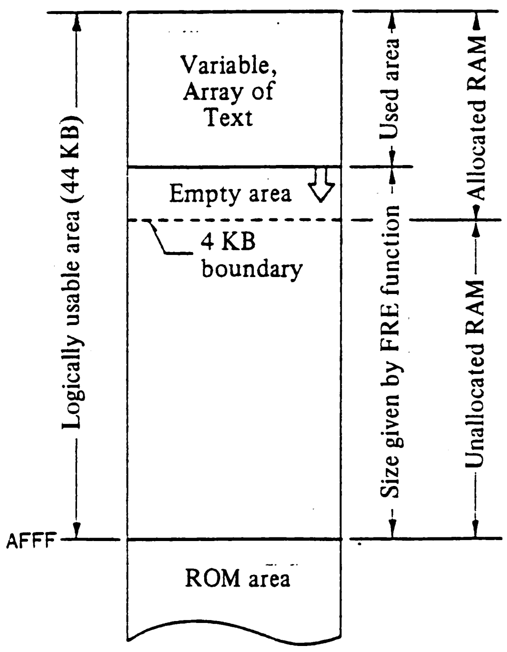
EXPLANATION

This statement gives the size of following unused area as a number of bytes according to the specified value of <argument>.

<argument>	Contents
0	Size of unused area in a variable or array area.
1	Size of unused area in a text area.
2	Size of actual RAM memory that has not been allocated.
Character expression	Size of unused area in a character area (range defined by CLEAR statement, 4 KB when initialized). Garbage collection (Note 1) is conducted when a character expression is specified for the <argument>.

(Note) Garbage collection: When a number of character expressions are defined, wasted areas are generated. The term "garbage collection" applies to rearranging these wasted areas to make a usable area.

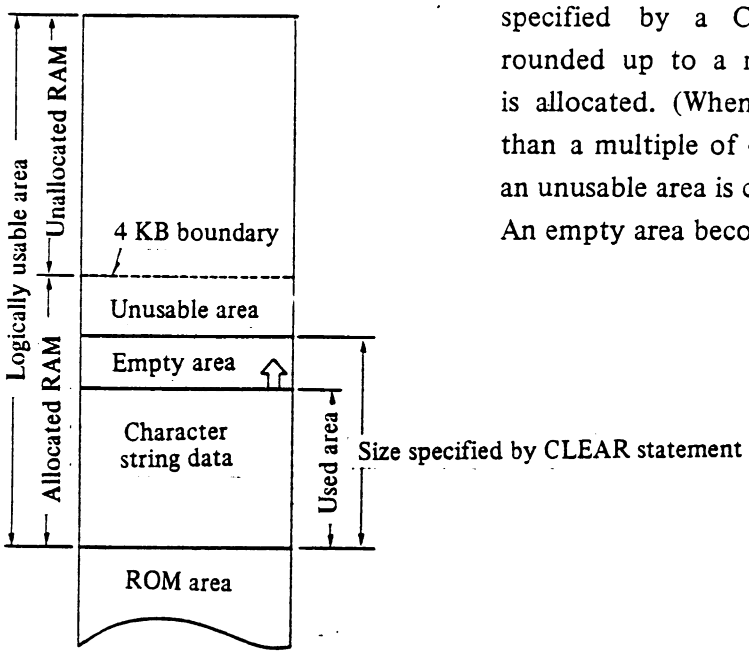
(1) Unused area of variable, array or text



For a variable, array or text, the BASIC interpreter allocates the necessary area from an unallocated area of RAM in 4-KB units.

- If there remains an unallocated part of RAM in the logically usable area and this part can be allocated in the actual RAM, the unused area is given in the form of:
  Empty area + RAM unallocated part
- An unused area that cannot be allocated is given as an "empty area".

(2) Unused area of character area



For the character area, RAM in a size specified by a CLEAR statement and rounded up to a multiple of 4096 bytes is allocated. (When a size that is smaller than a multiple of 4096 bytes is specified, an unusable area is created.)
An empty area becomes an unused area.

SAMPLE PROGRAM

```
Ready
LIST

100 ' FREE function
110 PRINT FRE(0),FRE(1),FRE(2),FRE("")
120 A$="ABC"
130 PRINT FRE(0),FRE(1),FRE(2),FRE("")
140 END

Ready
RUN
 28672          28558          24576          4094
 28667          28558          24576          4091

Ready
```

This is an example of a program in which Lines 110 and 130 display the size of the unused part of each area as well as displaying the size of unallocated actual RAM.

H

HEX$ function

FUNCTION — Gives a numeric value using a hexadecimal character string.

FORMAT — HEX$ (<argument>)

STATEMENT EXAMPLE — A$ = HEX$ (10)

EXPLANATION

This statement converts the specified <argument> to a hexadecimal character string. Specify an integer in the range 0 ~ 65535 for the <argument>.

REFERENCE

OCT$ function

SAMPLE PROGRAM

```
LIST

100 'HEX$ function
110 PRINT"DECIMAL     ";
120 FOR I=0 TO 15
130 PRINT USING "###";I;
140 NEXT I
150 PRINT
160 PRINT"HEXA DECIMAL ";
170 FOR J=0 TO 15
180 A$=HEX$(J)
190 PRINT USING"& &";A$;
200 NEXT J
210 END

Ready
RUN
DECIMAL       0 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
HEXA DECIMAL  0 1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
Ready
```

This program displays the conversion of decimal numbers, from 0 to 15, to hexadecimal numbers. The FOR ~ NEXT statements from Line 120 to Line 140 display decimal numbers 0 to 15, and the FOR ~ NEXT statements from Line 170 to Line 200 display hexadecimal numbers corresponding to decimal zero to 15.

INKEY$ function

FUNCTION	Gives one character from the keyboard buffer.
FORMAT	INKEY$
STATEMENT EXAMPLE	A$ = INKEY$

EXPLANATION

When this statement is given, the computer outputs the character of the pressed key and a null string for parts where no key is pressed. Since S1 BASIC has a buffer for the keyboard, when a key is pressed, it is stored in the buffer and read out from the buffer as needed. The INKEY$ function causes the output of a null string when the buffer is empty and a character if the buffer has a character in it.
This function is useful to have the computer conduct different processing when a key is pressed or no key is pressed. Use an INPUT$ function when waiting until a key input.
With the INKEY$ function, almost all keys except $\boxed{\text{BREAK}}$ and $\boxed{\text{CTRL}} + \boxed{\text{C}}$ can be used. The key pressed is not displayed on the screen.

REFERENCE

INPUT$ function

SAMPLE PROGRAM

```
Ready
LIST

100 'INKEY$ system parameter
110 PRINT"PRESS ANY KEY TO STOP PROGRAM"
120 A$ = INKEY$
130 IF A$ = "" GOTO 110 ELSE END

Ready
RUN
PRESS ANY KEY TO STOP PROGRAM
PRESS ANY KEY TO STOP PROGRAM
PRESS ANY KEY TO STOP PROGRAM
PRESS ANY KEY TO STOP PROGRAM
PRESS ANY KEY TO STOP PROGRAM
PRESS ANY KEY TO STOP PROGRAM
PRESS ANY KEY TO STOP PROGRAM
PRESS ANY KEY TO STOP PROGRAM
PRESS ANY KEY TO STOP PROGRAM
PRESS ANY KEY TO STOP PROGRAM

Ready
```

The INKEY$ function is useful to check whether or not a key has been pressed and what key has been pressed.
This program is executed until one of the keys is pressed.
Assign the results of the INKEY$ function operation to a character variable before it is used by other statements, as shown on Line 120.
The input is checked by the IF statement on Line 130 in when checking whether A$ is a null string or not.

INPUTS function

FUNCTION Inputs a character string from the keyboard or input file.

FORMAT INPUT$ (<number of characters> [, [#] <file No.>])

STATEMENT EXAMPLE INPUT$ (10, 1)

EXPLANATION

This statement reads the character string of the specified <number of characters> from the input file that is opened by the specified <file No.>.
Specify an integer in the range 0 ~ 255 for the <number of characters>. When "0" is specified as the <number of characters>, a null string is given. It can also be specified by a numeric expression.
If a character is input from the keyboard, it is not displayed on the screen but is input directly. However, this does not apply to pressing a key that temporarily stops or terminate the program being executed (BREAK key and others).
When the INPUTS function is used, the computer waits until the specified <number of characters> is input. If the buffer has a preceding input, that data is input.
Since most keys, except the BREAK key, CTRL + C keys and a few others, are effective with the INPTUS function, even the ⏎ key which cannot be input by a LINE INPUT statement, etc. can be input.

SAMPLE PROGRAM

```
Ready
LIST

100 'INPUT$ function
110 PRINT"ENTER CARRIAGE RETURN CODE TO STOP PROGRAM"
120 A$=INPUT$(1)
130 IF A$=CHR$(13) THEN 140 ELSE 110
140 END

Ready
RUN
ENTER CARRIAGE RETURN CODE TO STOP PROGRAM
ENTER CARRIAGE RETURN CODE TO STOP PROGRAM
ENTER CARRIAGE RETURN CODE TO STOP PROGRAM

Ready
```

The characteristic point of the INPUTS function is that it can read all control codes other than BREAK .
Execution of this program terminates if the input is carriage return, and the computer returns to the command level.

INSTR function (In string function)

FUNCTION Retrieves a character string.

FORMAT INSTR ([<argument>,] <character expression 1>, <character expression 2>)

STATEMENT EXAMPLE X = INSTR (5, B$, "A")

EXPLANATION

Specify the location of the starting character of <character expression 2> in the <character expression 1> counting from the starting character of <character expression 1>. Specify the position of search start for the <argument> with a number in the range 1 ~ 255.
When the <argument> is longer than the <character expression 1>, if the <character expression 1> is a null string or if <character expression 2> is not found in the <character expression 1>, "0" is given.
When the <character expression 2> is a null string, if the <character expression 1> is not a null string, the <argument> is given (default value is "1").

REFERENCE

LEN function

SAMPLE PROGRAM

```
Ready
LIST

100 'INSTR function
110 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
120 B$="N":C$="TU"
130 PRINT INSTR(A$,B$),INSTR(25,A$,C$)
140 END

Ready
RUN
 14              0

Ready
```

Line 110 assigns 26 alphabetic characters to A$. Also, B$ is set to "N" and C$ to "TU".
The first INSTR function on Line 130 returns the information about the location of the B$ contents to A$ counting from the start of A$. In this case, it is 14.
Since the <argument> is described as 25, the second INSTR function on Line 130 searches C$ starting at the 25th character of A$. However, since the C$ contents are at the 20th character from the top of A$, "0" is returned informing that there was no pertinent character string.

INT function (Integer function)

FUNCTION Changes to an integer.

FORMAT INT (<argument>)

STATEMENT EXAMPLE Y = INT (3.14)

EXPLANATION

This statement gives a maximum integer not exceeding the value of <argument>.

REFERENCE

FIX function, CINT function

NOTE

When the operation result of a floating point operation is specified as the <argument> of an INT function, the result may deviate as shown below affected by a calculation error.

```
LIST

10 A=43* .021* 2000/10+.4:PRINT A;INT(A);CDBL(A)
20 A=43*.0021*20000/10+.4:PRINT A;INT(A);CDBL(A)
30 A#=43*.0021#*20000/10+.4#:A=A#:PRINT A#;INT(A#);INT(A)

Ready
RUN
 181  181  181
 181  180  180.9999847412109
 181  180  181

Ready
```

In this example, the correct value is "181" on both lines of 10 and 20, but it is indicated as "180" on Line 20 since the result was slightly smaller than 181 affected by a calculation error and fractions are truncated.

Use one of the following methods to avoid this:

(1) Use the CINT function (regarding fractions of .5 over as a whole number and discarding the rest).

(2) Assign to a variable of integer type (the calculation of regarding fractions of .5 and over as whole numbers and discarding the rest is conducted at the time of assigning).

(3) Calculate in double precision, convert the result to a single precision value, and then use the INT function (refer to Line 30).

LEFT$ function

FUNCTION	Gives the left part of a character string.
FORMAT	LEFT$ (<character expression>, <argument>)
STATEMENT EXAMPLE	B$ = LEFT$ (A$, 3)

EXPLANATION

This statement gives a character string that consists of the same number of characters of the <argument> from the left end of the <character expression>. The <integer> is a number in the range 0 ~ 255. If "0" is specified for the <argument>, a null string is given. If the <argument> is greater than the number of characters of <character expression>, this functions gives the whole of <character expression>.

REFERENCE

MID$ function, RIGHT$ function

SAMPLE PROGRAM

```
Ready
LIST

100 'LEFT$ function
110 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
120 FOR I= 1 TO 26 STEP 2
130      B$=LEFT$(A$, I)
140      PRINT B$
150 NEXT I
160 END

Ready


RUN
A
ABC
ABCDE
ABCDEFG
ABCDEFGHI
ABCDEFGHIJK
ABCDEFGHIJKLM
ABCDEFGHIJKLMNO
ABCDEFGHIJKLMNOPQ
ABCDEFGHIJKLMNOPQRS
ABCDEFGHIJKLMNOPQRSTU
ABCDEFGHIJKLMNOPQRSTUVW
ABCDEFGHIJKLMNOPQRSTUVWXY

Ready
```

This is an example of operating character strings using the LEFT$ function.

Line 110 assigns 26 alphabetic to A$.

The FOR ~ NEXT statements starting at Line 120 display one character portion from the left of A$ using the LEFT$ function, while changing "I" from "1" to "26" in increment of 2.

LEN function (Length function)

FUNCTION	Gives the length of a character string.
FORMAT	LEN (<character expression>)
STATEMENT EXAMPLE	L = LEN (A$)

EXPLANATION

This statement gives the number of characters in a <character expression>. Even characters that are not output to the screen (from "0" to "31" character codes) are counted.

SAMPLE PROGRAM

```
Ready
LIST

100 'LEN function
110 PRINT"INPUT STRING AS LONG AS YOU LIKE."
120 INPUT A$
130 PRINT
140 PRINT"STRING LENGTH IS ";LEN(A$)
150 END

Ready
RUN
INPUT STRING AS LONG AS YOU LIKE.
? ABCDEFGHIJKLMNOPQRSTUVWXYZ

STRING LENGTH IS  26

Ready
```

This is an example of showing a character string length using the LEN function.
The INPUT statement on Line 120 inputs a character string from A to Z, and Line 140 shows the length using an LEN function.

LOF function (Length of file function)

FUNCTION	Gives the number of characters in an input buffer.
FORMAT	LOF (<file No.>)
STATEMENT EXAMPLE	B = LOF (1)

EXPLANATION

This statement checks the number of characters (0 ~ 255) contained in the input buffer specified by the <file No>.
This <file No> is the Buffer No. that is opened for input and it is not accompanied by 「#」.

SAMPLE PROGRAM

```
100 'LOF function
110 OPEN "O",#1,"test"
120 A$=STRING$(100,"?")
130 FOR I=1 TO 10
140 PRINT #1,A$
150 NEXT I
160 CLOSE #1
162 PRINT"ﾃｰﾌﾟ ｦ ﾏｷﾓﾄﾞｼﾃｸﾀﾞｻｲ !!!"
164 STOP
170 OPEN "I",#2,"test"
180 PRINT LOF(2)
230 CLOSE #2
250 END

Ready
RUN
ﾃｰﾌﾟ ｦ ﾏｷﾓﾄﾞｼﾃｸﾀﾞｻｲ !!!

Break In 164
Ready
CONT
 255

Ready
```

Lines 110 through 160 create a file consisting of 1,000 「?」 symbols. Line 170 opens the file and Line 180 prints out the value of LOF(2). At this time, when the message is output, the tape should be rewound.

LOG function

FUNCTION Gives a natural logarithm.

FORMAT LOG (<argument>)

STATEMENT EXAMPLE A = LOG (10)

EXPLANATION

This statement gives the natural logarithm of <argument>. The <argument> must be a positive value. The LOG function operation is conducted in single precision.

LOG (X) ≡ $\log_e X$

SAMPLE PROGRAM

```
Ready
LIST

100 'LOG function
110 FOR A=1 TO 10
120  PRINT LOG(A),
130 NEXT A
140 END

Ready
RUN
 0          .693147    1.09861    1.38629    1.60944
 1.79176    1.94591    2.07944    2.19722    2.30259

Ready
```

This program prints out natural logarithms for 1 ~ 10.

MAP function

FUNCTION	Conducts coordinate conversion between world coordinates and screen coordinates.
FORMAT	MAP (<coordinate>, <function>)
STATEMENT EXAMPLE	MAP (500, 0)

EXPLANATION

This statement converts the screen coordinates or world coordinates specified by the <coordinate> to the corresponding world coordinates or screen coordinates and gives the result as a function value. The conversion is conducted as shown below by the value (0 ~ 3) of <function>.

<function>	Contents
0	Sets the specified <coordinate> as the X coordinate of the world coordinates and converts it to the X coordinate of corresponding screen coordinates.
1	Sets the specified <coordinate> as the Y coordinate of the world coordinates and converts it to the Y coordinate of corresponding screen coordinates.
2	Sets the specified <coordinate> as the X coordinate of screen coordinates and converts it to the X coordinate of corresponding world coordinates.
3	Sets the specified <coordinate> as the Y coordinate of screen coordinates and converts it to the Y coordinate of corresponding world coordinates.

NOTE

Even if the value after the conversion is out of the corresponding view port or window, it does not create an error and the result is returned as it is.

SAMPLE PROGRAM

```
Ready
LIST

100 ' MAP function
110 SCREEN 0,0
120 WINDOW(10,10)-(50,50)
140 INPUT "ﾜｰﾙﾄﾞ ｻﾞﾋｮｳﾁ (X)=";X
142 INPUT "ﾜｰﾙﾄﾞ ｻﾞﾋｮｳﾁ (Y)=";Y
144 PRINT "ｽｸﾘｰﾝ ｻﾞﾋｮｳﾁ"
150 PRINT "X=";MAP(X,0)
160 PRINT "Y=";MAP(Y,1)
170 END

Ready
RUN
ﾜｰﾙﾄﾞ ｻﾞﾋｮｳﾁ (X)=? 1000
ﾜｰﾙﾄﾞ ｻﾞﾋｮｳﾁ (Y)=? 2000
ｽｸﾘｰﾝ ｻﾞﾋｮｳﾁ
X= 7895
Y= 9900

Ready
```

Lines 150 and 160 show the value of screen coordinates converted from world coordinates.

MID$ function

FUNCTION

Extracts a character string.

FORMAT

MID$ (<character expression>, <argument 1> [, <argument 2>])

STATEMENT EXAMPLE

A$ = MID$ (B$, 2, 3)

EXPLANATION

This statement extracts a character string consisting of the number of characters specified by <argument 2> starting at the character, the location of which is specified by <argument 1>, from the specified <character expression>. Specify an integer in the range 1 ~ 255 for <argument 1> and in the range 0 ~ 255 for <argument 2>. If <argument 2> is "0" or the <character expression> length is shorter than <argument 1>, a null string is given. When the specification of <argument 2> is omitted, or when the number of characters of <character expression> after the position specified by <argument 1> is less than the <argument 2> value, all characters after the position specified by <argument 1> are given.

REFERENCE

LEFT$ function, RIGHT$ function

SAMPLE PROGRAM

```
Ready
LIST

100 'MID$ function
110 INPUT "INPUT STRING:",A$
120 PRINT "PRESS ANY KEY FOR NEXT CHARACTER"
130 FOR C=1 TO LEN(A$)
140  PRINT MID$(A$,C,1);
150  Z$=INPUT$(1)
160 NEXT C
170 END

Ready
RUN
INPUT STRING:ABCDEF
PRESS ANY KEY FOR NEXT CHARACTER
ABCDEF
Ready
```

This program shows each character of the input character string. Line 110 inputs a character string. The FOR ~ NEXT loop on Lines 130 to 160 displays each character of the input character string. Line 140 shows one character at the "C"th position of A$, and Line 150 sets the computer to the state of waiting for key input. If a key is pressed, the next character is displayed.

MKI$, MKS$, MKD$ functions (Make integer dollar, single dollar, double dollar)

FUNCTION	Converts an integer to character string.
FORMAT	MKI$ (<argument>) MKS$ (<argument>) MKD$ (<argument>)
STATEMENT EXAMPLE	1 A$ = MKI$ (100) 2 B$ = MKS$ (3.14) 3 C$ = MKD$ (0.123456#)

EXPLANATION

The MKI$ function converts an integer to a 2-byte character string. The MKS$ function converts a single precision real number to a 4-byte character string. The MKD$ function converts a double precision real number to an 8-byte character string. Since these functions perform conversion while regarding the internal expression of each numeric value as a character code, they do not require as many bytes as the STR$ function.
The data converted by the MKI$, MKS$ and MKD$ functions can be converted back to the initial numeric values by the CVI, CVS and CVD functions.

REFERENCE

CVI, CVS, CVD functions

MREAD function (Mouse read function)

FUNCTION	Obtains the positional information of the mouse.
FORMAT	MREAD (<function>)
STATEMENT EXAMPLE	MREAD (0)

EXPLANATION

This statement gives the current screen coordinates or direction of the mouse as a function value depending on the value specified for the <function> (0 ~ 3).

<function>	Contents
0	Gives the X-coordinate of mouse.
1	Gives the Y-coordinate of the mouse.
2	Gives the direction of the X-coordinate of mouse. ("0" when in positive (right) direction and "−1" when in negative (left) direction).
3	Gives the direction of the Y-coordinate of the mouse. ("0" when in positive (downward) direction and "−1" when in negative (upward) direction).

All values of the MREAD function are reset when a RUN, NEW or CLEAR statement is executed.

SAMPLE PROGRAM

```
Ready
LIST

100 'MREAD function
120 ON MTRIG(1) GOSUB 500
130 MTRIG(1) ON
140 PRINT "ﾏｳｽ を いと゛うして, ﾄﾘｶ゛ ﾎ゛ﾀﾝ A を おしてくた゛さい。"
150 FOR I=1 TO 10000 :NEXT
160 END
500 '    Interruot routine
510 PRINT "X = "MREAD(0) ,"Y = "MREAD(1)
520 RETURN

Ready
RUN
ﾏｳｽ を いと゛うして, ﾄﾘｶ゛ ﾎ゛ﾀﾝ A を おしてくた゛さい。
X =  59         Y =  0
X =  93         Y =  118

Ready
```

This programs displays the X and Y coordinate values of the mouse on the screen.

MTRIG function (Mouse trigger function)

FUNCTION	Gives the state existing when a mouse trigger button is pressed.
FORMAT	MTRIG (<function>)
STATEMENT EXAMPLE	A = MTRIG (1)

EXPLANATION

This function obtains the information related to the mouse trigger button.
There are two types of information, one is the current state of the trigger button (level sense) and the other is information whether or not the trigger button has been pressed at least once (trigger sense). One of the following function values is given according to the value of <function>.

<function>	Contents
1	Regardless of the specification of MTRIG ON statement, "−1" is given when 「Button A」 of the mouse is being pressed, and "0" is given otherwise (level sense).
2	Regardless of the specification of MTRIG ON statement, "−1" is given when 「Button B」 of the mouse is being pressed, and "0" is given otherwise (level sense).
3	"−1" is given if 「Button A」 has been pressed at least once since execution of a 「MTRIG(1) ON」 or MTRIG(3) function, and "0" is given otherwise (trigger sense).
4	"−1" is given if 「Button B」 has been pressed at least once since execution of a 「MTRIG(2) ON」 or MTRIG(4) function, and "0" is given otherwise (trigger sense).

NOTE

Functions 3 and 4 give a function value once and are set to "0" until the button is pressed the next time.

SAMPLE PROGRAM

```
Ready
LIST

100 ' MTRIG Function
110 PRINT "ﾏｳｽ ﾎﾞﾀﾝ A ｶﾞ ｵｻﾚﾙﾏｴ ﾉ ｱﾀｲ !! ";MTRIG(3)
112 MTRIG(1) ON
120 PRINT "ﾏｳｽ ﾎﾞﾀﾝ A ｦ ｵｼﾃ ｸﾀﾞｻｲ !! "
130 A=MTRIG(3)
140 IF A=0 THEN 130
150 PRINT "ﾏｳｽ ﾎﾞﾀﾝ A ｶﾞ ｵｻﾚﾀﾄｷ ﾉ ｱﾀｲ !!! ";A
160 END

Ready
RUN
ﾏｳｽ ﾎﾞﾀﾝ A ｶﾞ ｵｻﾚﾙﾏｴ ﾉ ｱﾀｲ !!   0
ﾏｳｽ ﾎﾞﾀﾝ A ｦ ｵｼﾃ ｸﾀﾞｻｲ !!
ﾏｳｽ ﾎﾞﾀﾝ A ｶﾞ ｵｻﾚﾀﾄｷ ﾉ ｱﾀｲ !!! -1

Ready
```

This program terminates when Trigger button A of the mouse is pressed.

O

OCT$ function

FUNCTION	Gives an octal number of the numeric value in a character string form.
FORMAT	OCT$ (<argument>)
STATEMENT EXAMPLE	A$ = OCT$ (100)

EXPLANATION

This statement converts the <argument> to an octal number and gives the character string. Specify an integer in a range of 0 ~ 65535.

REFERENCE

HEX$ function

SAMPLE PROGRAM

```
Ready
LIST

100 'OCT$ function
110 INPUT "INPUT HEXA NUMBER:",H$
120 IF H$="" THEN GOTO 200
130 PRINT "&H";H$;"=&O";OCT$(VAL("&H"+H$))
140 GOTO 110
200 END

Ready
RUN
INPUT HEXA NUMBER:20
&H20=&O40
INPUT HEXA NUMBER:8
&H8=&O10
INPUT HEXA NUMBER:

Break In 110
Ready
```

This program converts a hexadecimal number to an octal number and displays it. Line 110 inputs a hexadecimal number character string to H$ and Line 130 prints out the octal number character string.

PEEK function

FUNCTION	Gives the contents of a memory address.
FORMAT	PEEK (<memory address>)
STATEMENT EXAMPLE	A = PEEK (&HC123)

EXPLANATION

This statement gives the contents of <memory address> as one byte. Specify the <memory address> in the form of an expression. The specification must be an integer in a range of 0 ~ 65535.
In the case of SI BASIC, data is read from the current machine language area. Switch the machine language area using a DEF MAP statement when necessary.

REFERENCE

POKE, DEF MAP

SAMPLE PROGRAM

```
Ready
LIST

100 ' PEEK function
110 CLEAR ,&H8000
130 FOR I%=0 TO  4
140 PRINT HEX$(PEEK(I%+&H9000))
150 NEXT I%
160 END

Ready
RUN
FF
FF
0
0
FF

Ready
```

Line 130, etc. read the 5-byte contents from &H9000 and display the result.

POINT function

FUNCTION	Gives a color to the dot displayed at the specified screen coordinates.
FORMAT	POINT (<Sx, Sy>)
STATEMENT EXAMPLE	A = POINT (50, 100)

EXPLANATION

This statement gives a color to the dot on the screen coordinates specified by <Sx, Sy> by a Palette No. When the specified value is outside of the view port, the function value is set to "−1".

SAMPLE PROGRAM

```
Ready
LIST

100 'POINT function
110 CLS
112 SCREEN 1,0,0
120 LINE(40,0)-(60,30),,PSET,7,B
130 A=POINT(40,0)
132 LOCATE 10,0
140 PRINT A
150 END

Ready
```

Line 140 displays the color of the dot at the upper left corner of the rectangle drawn by Line 120.

POS function (Position function)

FUNCTION	Gives the value of the horizontal position of cursor on the screen or the number of characters output to an output file.
FORMAT	POS (<argument>)
STATEMENT EXAMPLE	A = POS (0)

EXPLANATION

When "0" is specified for the <argument>, the function has a value that corresponds with the horizontal position of the cursor on the screen. The range of value that the function has at the time is 0 ~ 39 when the number of horizontal display characters is 40, and 0 ~ 79 when it is 80.
A File No. may be specified as the <argument>. The range of File No. is 1 ~ 16, and it must be the File No. of a file that is opened in the output mode. The table below shows the values of function on devices that can be opened in the output mode and cases in which the function value is set to "0". Also, when the value exceeds the output width specified by a WIDTH statement, "0" is set.

Device name	Function value	Case of function value set to "0"
Screen	The same as specifying "0" for <argument>	When the cursor is at the left edge of screen.
Printer	Number of characters output to file	(1) When return & line-feed codes are output. (2) "1" is set when the number of output characters exceeds 80.
RS-232C port (output mode)	Same as above	(1) When a character of code 13 (&HOD) is output. (2) When the number of output characters reaches 256.
Cassette (output mode)	Always "0"	Always.

REFERENCE

CSRLIN function, WIDTH

SAMPLE PROGRAM

```
Ready
LIST

100 ' POS function
110 PRINT "ABC";
120 A=POS(0)
130 PRINT:PRINT A
140 PRINT "ABC",
150 A=POS(0)
160 PRINT:PRINT A
170 END

Ready
RUN
ABC
 3
ABC
 14

Ready
```

The cursor is located at the third character when a semicollon ┌;┘ is used as a delimiter as on Line 110 and at the 14th character when a comma ┌,┘ is used as a delimiter as on Line 140.

RIGHT$ function

FUNCTION	Gives the right part of a character string.
FORMAT	RIGHT$ (<character expression>, <argument>)
STATEMENT EXAMPLE	C$ = RIGHT$ (A$, B + 2)

EXPLANATION

This statement gives a character string as the number of characters specified by the <argument> from the right side of <character expression>. Specify the <argument> as an integer in the range 0 ~ 255. When "0" is specified for the <argument>, the function gives it as a null string. If the number of characters in the <character expression> is larger than the <argument>, the whole <character expression> is given.

REFERENCE

LEFT$ function, MID$ function

SAMPLE PROGRAM

```
Ready
LIST

100 ' RIGHT$ function
110 A$="654321"
120 B$=RIGHT$(A$,4)
130 C$=RIGHT$(A$,4.512)
140 D$=RIGHT$(A$,10)
150 PRINT A$,B$,C$,D$
160 END

Ready
RUN
654321          4321            54321           654321

Ready
```

This program is an example of using the RIGHT$ function in many forms as shown on Lines 120 through 140.

RND function (Random function)

FUNCTION	Gives a random number.
FORMAT	RND [(<argument>)]
STATEMENT EXAMPLE	D = RND (1)

EXPLANATION

This statement generates a random number that is greater than "0" and smaller than "1". This function can be used in three ways as shown below by specifying a different value to the <argument>. The <argument> may be specified by a calculation expression or a variable to which a value set elsewhere is assigned..

(1) When the specification of <argument> is omitted or a positive value is specified, a random number system that is predetermined is generated and it is applied to the RND function in a program.

(2) When "0" is specified for the <argument>, a random number that is generated immediately before is applied.

(3) When a negative value is specified as the <argument>, a random number that is determined by the value of the argument is generated. This random number is not affected even if a random number system has been specified by a RANDOMIZE statement beforehand.

Thus, regardless of the specified argument, random numbers that can be obtained become that same each time the program is run. To avoid this, specify a different negative value as the <argument> each time a program is run. A different random number system can be obtained each time the program is run if a RANDOMIZE statement is used.

REFERENCE

RANDOMIZE

SAMPLE PROGRAM

```
Ready
LIST

100 ' RND function
110 FOR I=1 TO 5
120  PRINT RND(1),
130 NEXT I
140 PRINT:CLEAR
150 FOR I=1 TO 5
160  PRINT RND(2),
170 NEXT I
180 PRINT:CLEAR
190 A=RND(-1)
200 FOR I=1 TO 5
210  PRINT RND(1),
220 NEXT I
230 PRINT:CLEAR
240 FOR I=1 TO 5
250  PRINT RND(0),
260 NEXT I
270 PRINT
280 END

Ready

RUN
 .591065  .207991  .550967  .635863  .10411

 .591065  .207991  .550967  .635863  .10411

 .958935  .457458  .103187  .633296  .504862

 .311635  .311635  .311635  .311635  .311635


Ready
```

The random number system of Lines 110 ~ 140 is the same as that of Lines 150 ~ 180. A different random number system can be obtained if a RMD function with a negative value argument is used on Line 190. If "0" is used for the argument, as on Line 250, the same numbers are used as random numbers.

SCREEN function

FUNCTION	Gives the character code or attribute of a specified position on the text screen.
FORMAT	SCREEN (<X>,<Y> [,<select switch>])
STATEMENT EXAMPLE	SCREEN (20, 10, 1)

EXPLANATION

This function checks the character code or attribute of the display at a specified position on the text screen. The position can be specified in the same way as in the LOCATE statement. This function takes the character code if the specification of <select switch> is omitted or "0" is specified and the attribute if "1" is specified.
The attribute is 1-byte or data, and each bit has the following meaning.

0	S	I	G	R	Palette No.

$$R = \begin{cases} 0\text{: Normal} \\ 1\text{: Reverse} \end{cases} \qquad I = \begin{cases} 0\text{: Normal character} \\ 1\text{: IG character} \end{cases}$$

$$SG = \begin{cases} 00\text{: Displays text only.} \\ 01\text{: Displays graphics only.} \\ 10\text{: Text is given the priority in superimposition.} \\ 11\text{: Graphics are given the priority on superimposition.} \end{cases}$$

SAMPLE PROGRAM

```
LIST

100 ' SCREEN function
110 LOCATE 35,10: PRINT "A"
120 PRINT "&H";HEX$(SCREEN(35,10))
130 END

Ready
RUN


                                   A
&H41

Ready
```

In this program, the character written by Line 110 is read and output by Line 120.

SGN function (Sign function)

FUNCTION	Gives a sign.
FORMAT	SGN (<argument>)
STATEMENT EXAMPLE	S = SGN (A + B)

EXPLANATION

"1" is given if the <argument> is positive and "0" is given if the <argument> is "0". "–1" is given if the <argument> is negative.

SAMPLE PROGRAM

```
Ready
LIST

100 ' SGN function
110 DEF FNFABS(X)=(SGN(X)*X)
120 FOR I=-3 TO 3
130  PRINT FNFABS(I),
140 NEXT I
150 END

Ready
RUN
 3             2             1             0             1
 2             3
Ready
```

A function called "ABS" is provided as a standard function to obtain absolute values and this program defines it using the SGN function. Line 110 defines a function called "FNFABS" and Lines 120 through 140 print out the value of the FNFABS function by changing the argument value from "–3" to "3".

SIN function (Sine function)

FUNCTION — Gives a sine value.

FORMAT — SIN (<argument>)

STATEMENT EXAMPLE — PRINT SIN (A * B)

EXPLANATION

This statement gives the sine value of <argument>. The unit of <argument> is radians. The operation of the SIN function is conduced in single precision.

SIN (X) = sinX

REFERENCE

COS function, TAN function

SAMPLE PROGRAM

```
Ready
LIST

100 ' SIN function
110 PI=3.1415927#
120 FOR I=-.5 TO .5 STEP .1
130  PRINT SIN(PI*I),
140 NEXT I
150 END

Ready
RUN
-1             -.951057  -.809017  -.587785  -.309017
-4.68134E-08    .309017   .587785   .809017   .951057
 1
Ready
```

SPACE$ function

FUNCTION Gives a blank character string.

FORMAT SPACE$ (<argument>)

STATEMENT EXAMPLE A$ = SPACE$ (B + C)

EXPLANATION

This statement gives a character string consisting of a number of blanks that is equal to the specified <argument> value. Specify the <argument> with an integer in the range 0 ~ 255. If "0" is specified, a null string is given.

REFERENCE

SPC function

SAMPLE PROGRAM

```
Ready
LIST

100 ' SPACE$ function
110 FOR I=1 TO 5
120  A$=SPACE$(I):PRINT A$;STRING$(6-I,"*")
130 NEXT I
140 END

Ready
RUN
 *****
  ****
   ***
    **
     *

Ready
```

This program displays a number of ⌜*⌟ symbols in right justification by increasing one space at the left side for each decrease of ⌜*⌟ displayed.

SPC function (Space function)

FUNCTION Outputs blanks.

FORMAT SPC (<argument>)

STATEMENT EXAMPLE LPRINT SPC(N) ; A$; B$

EXPLANATION

This statement outputs blanks in the number of specified <argument>. A SPC function can only be used in a PRINT, LPRINT or PRINT# statement. The range of <argument> is 0 ~ 255.

REFERENCE

SPACE$ function

SAMPLE PROGRAM

```
Ready
LIST

100 ' SPC function
110 FOR I=1 TO 5
120  PRINT SPC(I);STRING$(6-I,"#")
130 NEXT I
140 END

Ready
RUN
 #####
  ####
   ###
    ##
     #

Ready
```

The result of this program execution is the same as that of the sample program for the SPACE$ function. It will be understood that the SPACE$ function and SPC function act in the similar ways.

SQR function (Square root function)

FUNCTION Gives a square root.

FORMAT SQR (<argument>)

STATEMENT EXAMPLE PRINT SQR (A + B * C)

EXPLANATION

This statement gives the square root of <argument>. The <argument> must be either "0" or a positive value. The SQR function operation is conducted in single precision.

SQR (X) $\equiv \sqrt{X}$

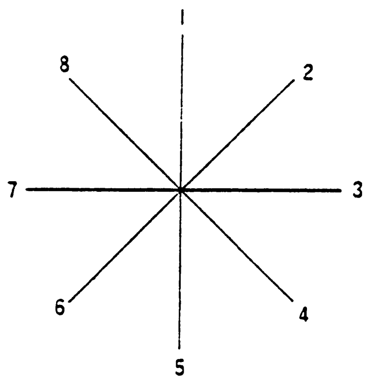
SAMPLE PROGRAM

```
Ready
LIST

100 ' SQR function
110 FOR N=1 TO 5
120  PRINT SQR(N),
130 NEXT N
140 END

Ready
RUN
 1             1.41421       1.73205       2             2.23607

Ready
```

The FOR ~ NEXT loop of lines 110 through 130 displays the values of square roots of 1 ~ 5.

STICK function

FUNCTION	Gives the direction in which the joystick is inclined.
FORMAT	STICK (n)
	n = 1 . . . . Joystick A
	n = 2 . . . . Joystick B
STATEMENT EXAMPLE	A = STICK (1)



EXPLANATION

Direction A of the joystick is shown as A = STICK(n). For example, when the following program is executed:

```
10 PRINT STICK (1)
20 GO TO 10
```

the numeric of the direction in which the joystick is inclined when the trigger button was pressed is displayed. "0" is displayed when it is upright.

SAMPLE PROGRAM

```
Ready
LIST

100 'STICK function
110 FOR I=1 TO 500
120 PRINT STICK(1);
130 NEXT I
140 END

Ready
RUN
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 0 0 0 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 0 0 0 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 0 0 0 0 0 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 0 0 0 0 0 0 0 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Ready
```

Line 120 displays the numeric of the direction in which the joystick is inclined when trigger button is pressed.

STRIG function (Joystick trigger function)

FUNCTION	Gives the state of a joystick trigger button when it is pressed.
FORMAT	STRIG (HfunctionK)
STATEMENT EXAMPLE	A = STRIG (1)

EXPLANATION

This function gives information related to joystick trigger buttons.
There are two types of information, one about the current state of the trigger button (level sense) and the other is whether or not the trigger button has been pressed at least once (trigger sense). The following value is given according to the value of <function>.

<function>	Contents
1	"−1" is given if the button of joystick A is currently being pressed. "0" is given otherwise. (level sense).
2	"−1" is given if the button of joystick B is currently being pressed. "0" is given otherwise. (level sense)
3	"−1" is given if the button of joystick A has been pressed at least one since execution of a 「STRIG (1) ON」 or STRIG (3) function. Otherwise, "0" is given. (trigger sense)
4	"−1" is given if the button of joystick B has been pressed at least once since execution of a 「STRIG (2) ON」 or STRIG (4) function. Otherwise, "0" is given. (trigger sense)

NOTE

Functions 3 and 4 give a function value once and then are set to "0" until the button is pressed for a second time.

SAMPLE PROGRAM

```
Ready
LIST

100 ' STRIG function
110 STRIG(1) ON
120 PRINT "3 ﾋﾞｮｳ ｲﾅｲﾆ ﾎﾞﾀﾝｦ ｵｼﾃｸﾀﾞｻｲ"
130 TO=TIME+3
140 IF TIME < TO THEN 140
150 PRINT STRIG(3)
160 PRINT STRIG(3)
170 STRIG(1) OFF
180 PRINT "3 ﾋﾞｮｳ ｲﾅｲﾆ ﾎﾞﾀﾝｦ ｵｼﾃｸﾀﾞｻｲ"
190 TO=TIME+3
200 IF TIME < TO THEN 200
210 PRINT STRIG(3)
220 END

Ready

RUN
3 ﾋﾞｮｳ ｲﾅｲﾆ ﾎﾞﾀﾝｦ ｵｼﾃｸﾀﾞｻｲ
-1
 0
3 ﾋﾞｮｳ ｲﾅｲﾆ ﾎﾞﾀﾝｦ ｵｼﾃｸﾀﾞｻｲ
 0

Ready
```

When trigger button A is pressed, Line 120 displays “–1” and Line 140 displays “0”.

STRINGS function

FUNCTION	Gives a character string which repeats one character.
FORMAT	STRING$ (<argument 1>, <argument 2>)
STATEMENT EXAMPLE	A$ = STRING$ (10, &H41)

EXPLANATION

Specify the <argument 2> as a character expression or character code. This function gives the starting character of <argument 2> if it is specified as a character expression. If it is specified as a character code, the character that corresponds to the character code is written in a character string for the number of characters specified by <argument 1>. The specification of <argument 1> and specification of <argument 2>, if it is given with a character code, must be an integer in the range 0 ~ 255.

REFERENCE

STR$ function

SAMPLE PROGRAM

```
Ready
LIST

100 ' STRING$ function
110 FOR I=1 TO 9 STEP 2
120  PRINT SPACE$(4-INT(I/2));
130  PRINT STRING$(I,"*")
140 NEXT I
150 END

Ready
RUN
    *
   ***
  *****
 *******
*********

Ready
```

This program draws a triangular graphic pattern. Line 130 draws one ｢*｣. Line 120 instructs the same thing as TAB (5-INT (I/†)).

STR$ function (String dollar function)

FUNCTION Converts a numeric value to a character string (numeric).

FORMAT STR$ (<argument>)

STATEMENT EXAMPLE A$ = STR$ (A)

EXPLANATION

This statement converts a numeric or variable specified by <argument> to a character string.

REFERENCE

STRINGS$ function, VAL function

SAMPLE PROGRAM

```
Ready
LIST

100 ' STR$ function
110 INPUT "INPUT ANY NUMBER:",A
120 A$=STR$(A)
130 B=VAL(A$)
140 PRINT A$,B
150 END

Ready
RUN
INPUT ANY NUMBER:123
 123           123

Ready
```

Note leading space

The STR$ function is the inverse function of VAL function. This program shows it clearly. It is an example of inputting "123" as "A".

TAB function

FUNCTION	Moves the cursor to a specified horizontal position.
FORMAT	TAB (<argument>)
STATEMENT EXAMPLE	PRINT "123", TAB(2), "456"

EXPLANATION

This statement moves the cursor, while outputting spaces, to the horizontal position specified by the <argument> (0 or greater). If the specified <argument> is greater than the number of characters displayed on one line of the screen, the remaining value after dividing the number of characters that can be displayed on one line is used. If the remaining value is smaller than the current cursor position, the same action as when specifying the value as the <argument> is performed on the next line.
If the horizontal coordinate of the cursor position immediately before outputting TAB is greater than the <argument> value, TAB is output from the beginning of next line.
The TAB function is used in PRINT, LPRINT and PRINT# statements.

REFERENCE

SPACE$ function, SPC function

SAMPLE PROGRAM

```
Ready
LIST

100 'TAB function
110 FOR I=1 TO 5
120  PRINT TAB(I);"*"
130 NEXT I
140 END

Ready
RUN
 *
  *
   *
    *
     *

Ready
```

This program displays ┌*┘ while increasing the horizontal tab by one.

TAN function (Tangent function)

FUNCTION	Gives a tangent.
FORMAT	TAN (<argument>)
STATEMENT EXAMPLE	A = TAN (10)

EXPLANATION

This statement gives the tangent of <argument>. The unit of <argument> is radians. The operation of the TAN function is conducted in single precision.

TAN (X) $\equiv$ tanX

REFERENCE

SIN function, COS function, ATN function

SAMPLE PROGRAM

```
Ready
LIST

100 'TAN function
110 PI=3.1415926#
120 FOR I=-PI/4 TO PI/4 STEP PI/4
130  PRINT TAN(I)
140 NEXT I
150 END

Ready
RUN
-1
 0
 1

Ready
```

This program shows the values of tan $(-\frac{\pi}{4})$, tan (0), and tan $(\frac{\pi}{4})$.

TIME function

FUNCTION	Gives the number of seconds from the built-in clock.
FORMAT	TIME
STATEMENT EXAMPLE	A = TIME

EXPLANATION

This statement gives the number of seconds based on counting "00:00:00" a.m. as "0" second. Since the value given by the TIME function is interlocked with the TIME$ value, set the time by assigning TIME$.

NOTE

Errors may occur on the time if a floppy disc is used frequently.

REFERENCE

DATA function, TIME$ (system variable)

SAMPLE PROGRAM

Refer to「TIME$」.

TIME$ (system variable)

FUNCTION Gives the time from the built-in clock.

FORMAT TIME$

STATEMENT EXAMPLE A$ = TIME$
TIME$ = "08:30:15"

EXPLANATION

This statement gives the current time as the following character string.
    HH:MM:SS
"HH" shows the hour (counting in the 24-hour system), "MM" the minutes and "SS" the seconds. A character string of the same format can be assigned to set the time. All of these must be set as two digits by adding "0" as the 10's digit if "HH", "MM" or "SS" is a single digit.

REFERENCE

DATE$ (system variable), TIME function

SAMPLE PROGRAM

```
Ready
LIST

100 ' TIME$ (system variable)
101 TIME$="00:10:00"
110 INPUT "ALARM TIME=? (HH:MM:SS)",AL$:WIDTH 40
120 WHILE AL$<> TIME$
130  LOCATE 12,16:PRINT TIME$:TIME
140 WEND
150 BEEP
160 END

Ready
RUN
ALARM TIME=? (HH:MM:SS)"00:12:00"

               00:12:00 720

Ready
```

This is an alarm program. When the time set by Line 110 is reached, the buzzer rings. Up to that time, the current time is displayed in the center of the screen with a 40-character width.

U

USR function (User function)

FUNCTION	Calls out a user defined machine language subroutine.
FORMAT	USR [<No.>] (<argument>)
STATEMENT EXAMPLE	N = USR1 (A)

EXPLANATION

This statement calls out a machine language program that was prepared by the user. At this time, the operation result of the specified <argument> value is given to the USR function. The <No.> is a value in the range 0 ~ 9, and it corresponds with the No. defined by a DEF USR statement. The default value of <No.> is "0".
The USR function has only one argument. When the USR function is called, accumulator A waits for a value that shows the form of the argument.

Value of A	Form of argument
2	Integer
3	Character
4	Single-precision type
8	Double-precision type

If the argument is numeric, index register X holds the start address of the floating point accumulator in which the argument is stored.

When the argument is of the integer type:

	←X
Upper 8 bits	
Lower 8 bits	

When the argument is of the single-precision type:

Exponent part	←X
Mantissa upper 8 bits	
Mantissa middle 8 bits	
Mantissa lower 8 bits	

When the argument is of the double-precision type:

Exponent part	←X
Mantissa highest 8 bits	
↑	
│	
│	
│	
↓	
Mantissa lower 8 bits	

When the argument is of the character type, Index register X has a 3-byte data address called a string descriptor.

Argument

Length of character string	←X
Upper part of stored address	
Lower part of stored address	

The value that is returned as the value of the USR function is in the same form as the value delivered as the argument. If it is numeric, it is returned as a numeric value to the stored position. It is a character string, a string descriptor is stored. For return, hexadecimal "39" (RTS in Assembly language) is used in machine language.

SAMPLE PROGRAM

Refer to the sample program of DEF USR statement.

VAL function (Value function)

FUNCTION Converts a character string to a numeric.

FORMAT VAL (<character expression>)

STATEMENT EXAMPLE B = VAL ("100")

EXPLANATION

This statement gives the numeric value expressed by the <character expression>. If the first character in the <character expression> is not 「+」, 「−」, 「&」 or a numeric, "0" is specified. When any character other than a numeric (including alphabetic characters of A through F in the case of hexadecimal notation) appears, all characters after that are ignored.

NOTE

Spaces used in the <character expression> are all ignored.

REFERENCE

STR$ function

SAMPLE PROGRAM

```
Ready
LIST

100 'VAL function
110 INPUT "INPUT HEXA NUMBER:",H$
120 IF H$="" THEN GOTO 200
130  PRINT "&H";H$"=";VAL("&H"+H$)
140 GOTO 110
200 END

Ready
RUN
INPUT HEXA NUMBER:A
&HA= 10
INPUT HEXA NUMBER:F
&HF= 15
INPUT HEXA NUMBER:

Break In 110
Ready
```

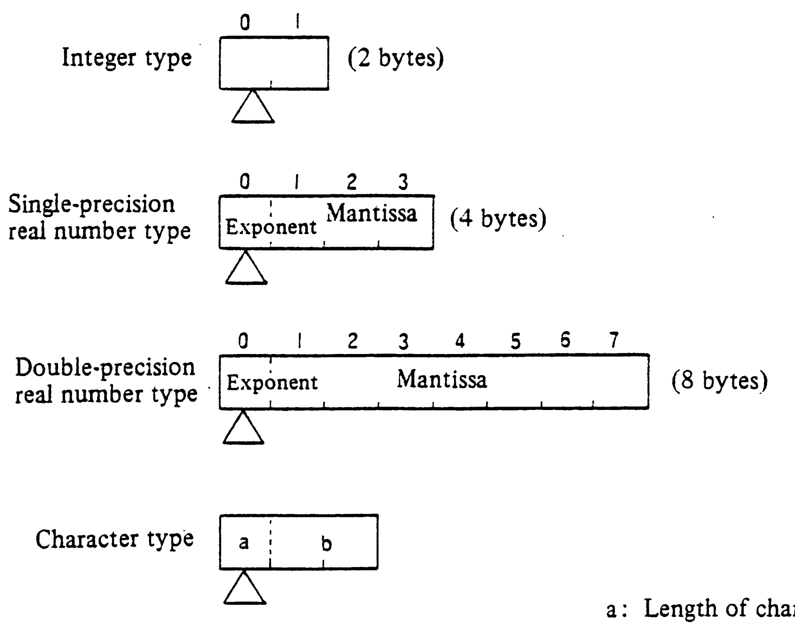
This is a hexadecimal-decimal conversion program. It terminates when a null string is input instead of a hexadecimal number.
Line 110 inputs a hexadecimal character to character variable H$, places character string &H before it, making it the argument of a VAL function (Line 130).

VARPTR function (Variable pointer function)

FUNCTION	Gives an address where variable data is stored.
FORMAT	VARPTR (<variable>)
STATEMENT EXAMPLE	A = VARPTR (B)

EXPLANATION

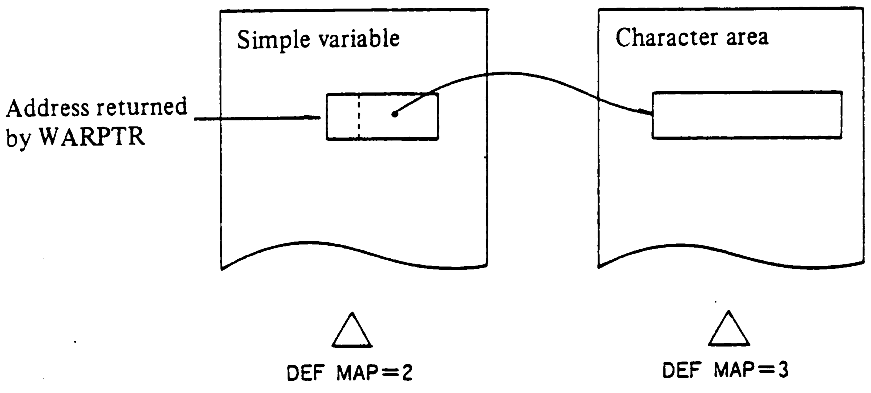
This statement gives the address where variable data or array element data is stored.
Specify the <variable> as a variable name or element of an array.
When the BARPTR function is used for an array, the value must have been assigned to all simple variables. This is because, if a value is assigned to a simple variable, the array is transferred to a later address.
Data is stored in the address shown by the VARPTR function as shown below.



a: Length of character string
b: Address of character string

NOTE

In the case of S1 BASIC, when using an address shown by BARPTR with a machine language function (PEEK and POKE instructions and others), first change the machine language area with a DEF MAP statement.
Especially, to use the address of a character type variable, a DEF MAP statement must be executed twice as shown below.



SAMPLE PROGRAM

```
Ready
LIST

100 ' VARPTR function
110 A(0)=0
120 PRINT HEX$(VARPTR(A(0)))
130 END

Ready
RUN
7

Ready
```

Line 110 defines the value of array variable A(0), and Line 120 obtains the address. Function HEXS on Line 120 is a function that converts the argument to a hexadecimal character string.

VIEW function

FUNCTION Gives a display coordinate value of a view port.

FORMAT VIEW (<argument>)

STATEMENT EXAMPLE A = VIEW (0)

EXPLANATION

This statement obtains the current view port position (Dx_1, Dx_2) – (Dx_2, Dy_2) that is specified by the display coordinates. The following coordinates, are given as the function value according to the <argument> value.

<argument>	Details
0	X-coordinate (Dx_1) of view port upper left
1	Y-coordinate (Dx_1) of view port upper left
2	X-coordinate (Dx_2) of view port lower right
3	Y-coordinate (Dx_2) of view port lower right

REFERENCE

VIEW statement, WINDOW function

SAMPLE PROGRAM

```
Ready
LIST

100 'VIEW function
110 SCREEN 0
130 VIEW(0,0)-(50,50)
140 PRINT"DX1= ";VIEW(0)
150 PRINT"DY1= ";VIEW(1)
160 PRINT"DX2= ";VIEW(2)
170 PRINT"DY2= ";VIEW(3)
180 END

Ready
RUN
DX1=  0
DY1=  0
DX2=  50
DY2=  50

Ready
```

Lines 140 through 170 show the view port position at the display coordinates specified by Line 130.

WINDOW function

FUNCTION	Gives a world coordinate value of a window.
FORMAT	WINDOW (<argument>)
STATEMENT EXAMPLE	A = WINDOW (1)

EXPLANATION

This statement obtains the window position (Wx_1, Wy_1) – (Wx_2, Wy_2) specified in the world coordinates. The following coordinates are given as the function value according to the <argument> value.

<argument>	Details
0	X-coordinate (Wx_1) of window upper left
1	Y-coordinate (Wy_1) of window upper left
2	X-coordinate (Wx_2) of window lower right
3	X-coordinate (Wy_2) of window lower right

REFERENCE

WINDOW statement, VIEW function

SAMPLE PROGRAM

```
Ready
LIST

100 'WINDOW function
110 SCREEN 1
120 PRINT "WX1= ";WINDOW(0)
130 PRINT "WY1= ";WINDOW(1)
140 PRINT "WX2= ";WINDOW(2)
150 PRINT "WY2= ";WINDOW(3)
160 WINDOW(10,10)-(50,50)
170 PRINT "WX1= ";WINDOW(0)
180 PRINT "WY1= ";WINDOW(1)
190 PRINT "WX2= ";WINDOW(2)
200 PRINT "WY2= ";WINDOW(3)
210 END

Ready
RUN
WX1=  0
WY1=  0
WX2=  639
WY2=  199
WX1=  10
WY1=  10
WX2=  50
WY2=  50

Ready
```

First, Lines 120 through 150 show the initial state of the window coordinates on the screen, and Lines 170 through 200 show the X and Y coordinate values specified by Line 160.

APPENDIX

1. SAMPLE PROGRAMS

This section shows sample programs contained in the table below.

Sample program	Details	Necessary peripheral devices
1	Graphic display	–
2	Key input control	–
3	Table drawing	–
4	RS-232C	RS-232C line
5	"	"
6	"	"
7	PF key interrupt	–
8	"	–
9	Mouse	Mouse
10	File input/output	Cassette tape recorder
11	Machine language user function	–

Programs 1, 2 and 3 are prepared to show beginners what BASIC can do. We hope that these will be helpful for you to write your own programs.
Programs 4 through 11 are prepared to help you understand explanations given in this manual.

Sample Program 1 – Graphic display –

This program displays a colored graphic pattern. We hope that you will enjoy it.

```
10 REM ***** ｸﾞﾗﾌｨｯｸ *****
20 CLS:K=1
30 FOR I=0 TO 639 STEP 8
40 C=C+1:IF C=8 THEN C=1                  ......... Sets colors
50 FOR J=0 TO 7
60 ON K GOSUB 100,120
70 NEXT J,I
80 IF K=1 THEN K=2:GOTO 30
90 END
100 LINE (0,0)-(I+J,199),PSET,C           ......... Draws a line
110 RETURN
120 LINE (639,199)-(I+J,0),PSET,C ......... Draws a line
130 RETURN
```

Sample Program 2 – Key input control –

This is an example of a program that can be used for a game. When 「•」 is displayed, move the cursor using the $\boxed{\leftarrow}$, $\boxed{\rightarrow}$, $\boxed{\uparrow}$, and $\boxed{\downarrow}$ keys and erase 「*」 in the surrounding area. The color changes when 「•」 is moved. Erase all 「*」 marks and input "0". The number of seconds spent is displayed at the top left corner of the screen.
Input "0" to stop in the middle.

```
10 WIDTH 40:SCREEN 1,,1
20 FOR I=1 TO 100
30 X=INT(RND*50):Y=INT(RND*50)          }
32 LOCATE X,Y:IF Y<24 THEN PRINT"*";    } Displays 「*」
40 NEXT                                 }
50 TIME$="00:00:00"                      ......... Sets the time
60 X=20:Y=10:LOCATE X,Y,1:PRINT"•"       ......... First display
70 A$=INPUT$(1)                          }
80 IF A$="0" THEN 160                    }
90 I=ASC(A$):IF I<28 THEN 70             } Judgement of characters that
100 ON I-27 GOTO 120,130,140,150         } are input by keys
110 GOTO 70                              }
120 GOSUB 190:X=X+1:GOSUB 200            }
130 GOSUB 190:X=X-1:GOSUB 200            } Moves 「•」
140 GOSUB 190:Y=Y-1:GOSUB 200            }
150 GOSUB 190:Y=Y+1:GOSUB 200            }
160 COLOR 7:LOCATE 0,0,3                 }
170 PRINT"TIME";TIME;"sec."              } Termination processing
180 IF INKEY$<>"" THEN 180 ELSE END      }
190 LOCATE X,Y:PRINT" ";:RETURN          ......... Erases preceding 「•」
200 '--- SUBROUTINE ---                  }
210 IF X<0 THEN X=39:GOTO 230            }
220 IF X>39 THEN X=0                     } Processing when drawn
230 IF Y<0 THEN Y=23:GOTO 250            } outside the screen
240 IF Y>23 THEN Y=0                     }
250 LOCATE X,Y:R=RND*6+1:COLOR R         }
260 PRINT"•";                            } Re-display of 「•」
270 RETURN 70                            }
```

Sample Program 3 – Writing a table –

This is an example of a program that writes a table. It draws the frame of the table using a LINE statement and writes the contents by a PRINT USING statement at the position specified by a LOCATE statement.

```
10 REM ***** ひよう *****
20 WIDTH 40
30 FOR I=0 TO 639 STEP 159
40 LINE (I,12)-(I,140),PSET,7,B
50 NEXT                                                      Displays a frame
60 FOR I=12 TO 150 STEP 16
70 LINE (0,I)-(635,I),PSET,7,B
80 NEXT
90 FOR I=1 TO 6                                              Reads data
100 READ A(I),B(I):NEXT
110 LOCATE 4,2:PRINT "I"
120 LOCATE 13,2:PRINT "A(I)"                                 Displays items
130 LOCATE 23,2:PRINT "B(I)"
140 LOCATE 32,2:PRINT "TOTAL"
150 J=2
160 FOR I=1 TO 6
170 J=J+2
180 LOCATE 3,J:PRINT I
190 LOCATE 12,J:PRINT USING "###.###";A(I)
200 LOCATE 22,J:PRINT USING "###.###";B(I)
210 LOCATE 32,J:PRINT USING "###.###";A(I)+B(I)
220 A=A(I)+A                                                 Displays values
230 B=B(I)+B
240 NEXT
250 LOCATE 2,16:PRINT "TOTAL"
260 LOCATE 12,16:PRINT USING "###.###";A
270 LOCATE 22,16:PRINT USING "###.###";B
280 LOCATE 32,16:PRINT USING "###.###";A+B
290 END
300 DATA 10.234,-29.43,-34.4,54.5
310 DATA -0.234,12.043,-15.4,14.65                           Data
320 DATA 9.568,10.1,-15.458,-.65
```

Sample Program 4 – Data output to RS-232C –

This program outputs data to RS-232C Port 0.

```
10 '*** OUTPUT TO RS-232C ***
20 OPEN "O",#1,"COM0:(S8N2)"
30 FOR J=1 TO 10
40 PRINT #1,"1234567890";
50 NEXT J
60 PRINT #1
70 CLOSE 1
80 END
```

Sample programs 4 and 5 and sample programs 4 and 6 make pairs. Combine them when executing them.

Sample Program 5 – Data input from the RS-232C Port –

This program inputs data from the RS-232C port.

```
10 REM *** INPUT FROM RS-232C ***
20 ON ERROR GOTO 100
30 OPEN "I",#2,"COM0:(S8N2)"
40 FOR J=1 TO 10
50 INPUT #2,A$
60 PRINT A$
70 NEXT J
80 CLOSE #2
90 END
100 IF ERR<>53 OR ERL<>50 OR J<>1 THEN ON ERROR GOTO 0
110 PRINT "このフ゜ロク゛ラムを し゛っこうするには RS-232Cホ゜ート0へ"
120 PRINT "にゅうりょくそうちか゛ つなか゛れてなけれは゛ なりません。"
130 J=10:RESUME 70
```

To execute this program, an input device must be connected to RS-232C Port 0.

Sample Program 6 – Control of interrupt from RS-232C –

Execution of this program displays the current time, minutes and seconds on the upper right corner of the screen.

```
10 REM *** INTERRUPT FROM RS-232C ***
20 ON KEY(3) GOSUB 210 :KEY(3) ON
30 OPEN "I",#2,"COM0:(S8N2)"
40 ON COM(0) GOSUB 100
50 WIDTH 40 :COM(0) ON
60 FOR J=1 TO 200 :NEXT
70 LOCATE 0,0 :PRINT SPC(23);
80 LOCATE 30,0 :PRINT TIME$
90 GOTO 80
100 REM *** COM(0) INTERRUPT PROCEDURE ***
110 LOCATE 0,0 :PRINT "COM(0) INTERRUPT BY (";
120 A$=INPUT$(1,#2)
130 PRINT A$;")";
140 RETURN 60
200 REM *** END ***
210 KEY(3) OFF
220 IF INKEY$="" THEN CLOSE:END ELSE 220
```

When execution of this program is stopped by pressing the BREAK key, etc., execute a CLOSE statement and close the file.
When an input is made from RS-232C Port 0, the execution jumps to Line 100 defined by Line 40. Line 120 fetches a character from the buffer and Line 130 displays it on the screen. RETURN 60 on Line 140 causes return of the execution to Line 60, and the computer waits for an input again. Press the PF3 key to terminate execution of this program.

Sample Program 7 – Control of PF key interrupt –

This is an example of controlling PF key interrupts.
Pressing the $\boxed{\text{PF1}}$ key generates an interrupt. A message "KEY (1) INTERRUPT" is displayed on the screen by the interrupt control routine. $\boxed{\text{PF3}}$ terminates execution.

```
10 REM *** KEY INTERRUPT ***
20 SCREEN 0 :WIDTH 40
30 ON KEY(1) GOSUB 100
40 ON KEY(3) GOSUB 200
50 KEY(1) ON :KEY(3) ON
60 FOR J=1 TO 200 :NEXT
70 LOCATE 0,0 :PRINT SPC(20);
80 LOCATE 30,0 :PRINT TIME$
90 GOTO 80
100 REM *** KEY(1) INTERRUPT PROCEDURE ***
110 COLOR 2
120 LOCATE 0,0 :PRINT "KEY(1) INTERRUPT "
130 COLOR 7
140 RETURN 60
200 REM *** END ***
210 KEY(1) OFF :KEY(3) OFF
220 IF INKEY$="" THEN END ELSE 220
```

When execution of this program is stopped by pressing the $\boxed{\text{BREAK}}$ key, etc., execute ｢KEY (1) OFF: KEY (3) OFF｣ and reset the key interrupt.

Sample Program 8 – Control of PF key interrupt –

This sample program is to generate a PF key interrupt, as in Sample Program 7. Pressing the cursor movement key moves the dot and when $\boxed{\text{PF1}}$ is pressed, a line is drawn up to the point where the cursor has moved. When $\boxed{\text{PF4}}$ is pressed, a line that is same as that drawn is displayed at the position of the displaced coordinates. When $\boxed{\text{PF5}}$ is pressed, the data on the screen is output to a cassette tape. $\boxed{\text{PF3}}$ terminates the execution of the program Line 10000 and lines after that make a routine for inputting the screen data from a cassette tape. Execute this routine using GOTO10000 if there is data on the cassette tape.

```
10 'KEY IRQ(2)
20 DEFINT X,Y,I,J
30 DIM X(2000),Y(2000):WIDTH 80:SCREEN 1
40 ON KEY(1) GOSUB 1000
50 ON KEY(3) GOSUB 4000
60 ON KEY(4) GOSUB 2000
70 ON KEY(5) GOSUB 3000
80 KEY(1) ON:KEY(3) ON:KEY(4) ON:KEY(5) ON
90 X0=320:Y0=100:X(I)=X0:Y(I)=Y0:I=1:PSET(X0,Y0),7
100 A$=INKEY$:IF A$="" GOTO 100
110 PRESET(X0,Y0)
120 IF A$=CHR$(&H1D) THEN X0=X0-1:GOTO 200
130 IF A$=CHR$(&H1C) THEN X0=X0+1:GOTO 200
140 IF A$=CHR$(&H1E) THEN Y0=Y0-1:GOTO 200
150 IF A$=CHR$(&H1F) THEN Y0=Y0+1
200 PSET(X0,Y0),7
210 GOTO 100
1000 X(I)=X0:Y(I)=Y0
1010 LINE(X(I-1),Y(I-1))-(X0,Y0),PSET,7
1030 I=I+1
1040 RETURN 100
2000 FOR J=1 TO I-1
2010 LINE(X(J-1)+10,Y(J-1)+5)-(X(J)+10,Y(J)+5),PSET,2
2020 NEXT
2030 RETURN 100
3000 OPEN "O",#1,"CAS0:SAMPLE"
3010 FOR J=0 TO I-1
3020 PRINT #1,X(J),Y(J)
3030 NEXT
3040 CLOSE
4000 KEY(1) OFF:KEY(3) OFF:KEY(4) OFF:KEY(5) OFF
4010 END
10000 OPEN "I",#1,"CAS0:SAMPLE"
10010 INPUT #1,X0,Y0
10020 FOR J=1 TO 1000
10030 IF EOF(1) GOTO 10080
10040 INPUT #1,X,Y
10050 LINE(X0,Y0)-(X,Y),PSET,1
10060 X0=X:Y0=Y
10070 NEXT
10080 CLOSE 1:END
```

When execution of this program is stopped by pressing the $\boxed{\text{BREAK}}$ key, etc., execute ⌜FOR I = 1 TO 5 : KEY (I) OFF : NEXT ⌞⌟ when waiting for a command.

Sample Program 9 – Mouse –

This sample program draws a simple picture using the mouse. It inputs the colors of lines and paint first. Moving the mouse moves the graphic cursor on the screen, and when Button A of the mouse is pressed, a line is drawn. Pressing PF3 paints the color. Press PF4 to draw another line. The line and paint colors can be changed by pressing PF1 and PF2 .

```
10 ' GCURSOR program
20 GOSUB 170
30 GOSUB 180
40 ON KEY(1) GOSUB 170 :KEY(1) ON
50 ON KEY(2) GOSUB 180 :KEY(2) ON
60 ON KEY(3) GOSUB 200:KEY(3) ON
70 ON KEY(4) GOSUB 190:KEY(4) ON
80 WIDTH80 :SCREEN 1,,0 :WINDOW(0,0)-(639,199) :VIEW(0,0)-(639,199)
90 CLS
100 X1=320 :Y1=100
110 GCURSOR(X1,Y1),(X2,Y2),"M"
120 LINE (X1,Y1)-(X2,Y2),PSET,CL
130 X1=X2:Y1=Y2:GOTO 110
140 GCURSOR(X2,Y2),(X2,Y2),"M"
150 PAINT(X2,Y2),CP,CL
160 GOTO 140
170 INPUT "LINE  の いろ = ",CL:CLS 2 :RETURN
180 INPUT "PAINT の いろ = ",CP:CLS 2 :RETURN
190 RETURN 100
200 RETURN 140
```

Sample Program 10 – File input/output –

This sample program registers an arbitrary character string, outputs it to a printer or cassette tape, or reads it from cassette tape.

At first, 「Load name file? (Y/N)」 is displayed. Input 「N」. Then the screen display is erased. Input a character string. When the following character string is input at this time, the functions shown below are performed.

SAVE :	Outputs data to a cassette tape.
PRINTER:	Outputs data to a printer.
END :	Terminates program execution and the computer is set to wait for a BASIC command.
DELETE .	Deletes the last input character string.

When the first 「Load name file? (Y/N)」 message is displayed on the second or later execution, if 「Y」 is input, the data can be read from the cassette tape.

```
100 '*** FILE INPUT & OUTPUT SAMPLE ***
110 CLEAR 1000: DEV$="CASO:"
120 DIM A$(200)
130 OPEN "I",#5,"KYBD:"
140 OPEN "O",#3,"LPTO:"
150 INPUT "Load name file ? (Y/N)",B$
160 IF B$="Y" THEN 500 ELSE IF B$<>"N" THEN 150
170 I=0: CLS
180 LINE INPUT #5,"NAME=",A$(I)
190 IF A$(I)="SAVE" THEN I=I-1: GOTO 400
200 IF A$(I)="PRINTER" THEN I=I-1: GOTO 300
210 IF A$(I)="END" THEN CLOSE: END
220 IF A$(I)<>"DELETE" THEN 240
230 PRINT CHR$(13);STRING$(2,30);CHR$(26);: I=I-2
240 IF I>198 THEN PRINT "Data full  !!": GOTO 180
250 I=I+1: GOTO 180
300 ' PRINT TO PRINTER
310 FOR J=0 TO I
320   PRINT #3,"NAME=";A$(J)
330 NEXT
340 GOTO 240
400 ' SAVE TO CASSETTE
410 GOSUB 600
420 OPEN "O",#2,DEV$+"NAMES": CLS
430 FOR J=0 TO I
440   PRINT #2,A$(J)
450   PRINT "NAME=";A$(J)
460 NEXT
470 CLOSE #2
480 I=I+1: GOTO 180
500 ' INPUT FROM CASSETTE
510 GOSUB 600: CLS
520 OPEN "I",#7,DEV$+"NAMES"
530 I=0
540 LINE INPUT #7,A$(I)
550 PRINT "NAME=";A$(I)
560 I=I+1
570 IF EOF(7) THEN CLOSE #7: GOTO 180 ELSE 540
600 ' SUBROUTINE
610 PRINT "Ready Cassette ?";INPUT$(1)
620 RETURN
```

Sample Program 11 – Machine language user function –

This sample program writes a machine language program using BASIC and calculates using machine language user functions. DEFMAP=0 on Line 30 defines the machine language program area. (Refer to the DEFMAP statement.)

```
10 '  USR function
20 CLEAR 1000,&H9FFF
30 DEFMAP=0
40 DEFUSR0=&HA000
50 FOR I=&HA000 TO &HA015
60 READ D$                                  Writes a machine language program
70 POKE I,VAL("&H"+D$)
80 NEXT I
90 INPUT "-16384 から 16383 の せいすう";B
100 IF B<-16384 OR B>16383 THEN 90
110 B%=B :A%=USR0(B%)
120 PRINT B%;"* 2=";A%
130 END
140 REM --- DATA --- ORG $A000
150 DATA F6,EF,59
160 DATA 58
170 DATA F7,EF,59
180 DATA F6,EF,58
190 DATA 25,05                              Data of machine language program
200 DATA 58
210 DATA F7,EF,58
220 DATA 39
230 DATA 59
240 DATA F7,EF,58
250 DATA 39
```

2. CHARACTER CODE TABLE

A character code expresses a character, and each character code corresponds to a character in the interlace or non-interlace mode. Character codes are expressed in hexadecimal notation 「&H <X value> <Y value>」. For example, the character code for 「A」 is X=4, Y=1 and it is expressed as 「&H41」. In BASIC, a hexadecimal number can be used as it is but with 「&H」 written at the start, and to find out the value in decimal notation, key input 「PRINT &H41」 when waiting for a command. Note that the character displayed on the screen for the same character code is different depending on the screen mode (interlace mode or non-interlace mode). Also note that, when X=0 or 1, the character is not output to the screen. In the case of BASIC, since 「&H7F」 is used as the code for DEL, 「\」 is not output. Also, 「&H15」 is used as the INS code.

In interlace mode

→ Upper 4 bits

↓ Lower 4 bits

Y \ X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		DE		0	@	P		p	年	π	〒	ー	タ	ミ	た	み
1	SH	D1	!	1	A	Q	a	q	月	あ	。	ア	チ	ム	ち	む
2	SX	D2	"	2	B	R	b	r	日	い	「	イ	ツ	メ	つ	め
3	EX	D3	#	3	C	S	c	s	市	う	」	ウ	テ	モ	て	も
4	ET	D4	S	4	D	T	d	t	区	え	、	エ	ト	ヤ	と	や
5	EQ	NK	%	5	E	U	e	u	町	お	・	オ	ナ	ユ	な	ゆ
6	AK	SN	&	6	F	V	f	v	を	か	ヲ	カ	ニ	ヨ	に	よ
7	BL	EB	'	7	G	W	g	w	ぁ	き	ァ	キ	ヌ	ラ	ぬ	ら
8	BS	CN	(	8	H	X	h	x	ぃ	く	ィ	ク	ネ	リ	ね	り
9	HT	EM	)	9	I	Y	i	y	ぅ	け	ゥ	ケ	ノ	ル	の	る
A	LF	SB	*	:	J	Z	j	z	ぇ	こ	ェ	コ	ハ	レ	は	れ
B	HM	EC	+	;	K	[	k	{	ぉ	さ	ォ	サ	ヒ	ロ	ひ	ろ
C	CL	→	,	<	L	¥	l	|	ゃ	し	ャ	シ	フ	ワ	ふ	わ
D	CR	←	−	=	M	]	m	}	ゅ	す	ュ	ス	ヘ	ン	へ	ん
E	SO	↑	.	>	N	^	n	~	ょ	せ	ョ	セ	ホ	゛	ほ	▩
F	SI	↓	/	?	O	_	o	\	っ	そ	ッ	ソ	マ	゜	ま	

In non-interlace mode

→ Upper 4 bits

Lower 4 bits ↓

Y \ X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		D_E		0	@	P		p	▁	┴	〒	ー	タ	ミ	═	╳
1	S_H	D_1	!	1	A	Q	a	q	▂	┬	。	ア	チ	ム	╞	円
2	S_X	D_2	"	2	B	R	b	r	▃	┤	「	イ	ツ	メ	╪	年
3	E_X	D_3	#	3	C	S	c	s	▄	├	」	ウ	テ	モ	╡	月
4	E_T	D_4	$	4	D	T	d	t	▅	▔	、	エ	ト	ヤ	◢	日
5	E_Q	N_K	%	5	E	U	e	u	▆	─	・	オ	ナ	ユ	◣	時
6	A_K	S_N	&	6	F	V	f	v	▇	│	ヲ	カ	ニ	ヨ	◥	分
7	B_L	E_B	'	7	G	W	g	w	█	▕	ァ	キ	ヌ	ラ	◤	秒
8	B_S	C_N	(	8	H	X	h	x	▏	┌	ィ	ク	ネ	リ	♠	┼
9	H_T	E_M	)	9	I	Y	i	y	▎	┐	ゥ	ケ	ノ	ル	♥	⋎
A	L_F	S_B	*	:	J	Z	j	z	▍	└	ェ	コ	ハ	レ	♦	⋏
B	H_M	E_C	+	;	K	[	k	{	▌	┘	ォ	サ	ヒ	ロ	♣	♪
C	C_L	→	,	<	L	¥	l	|	▋	╭	ャ	シ	フ	ワ	●	÷
D	C_R	←	−	=	M	]	m	}	▊	╮	ュ	ス	ヘ	ン	○	±
E	S_O	↑	.	>	N	^	n	~	▉	╰	ョ	セ	ホ	゛	╱	▒
F	S_I	↓	/	?	O	_	o	\	┼	╯	ッ	ソ	マ	゜	╲	

3. ERROR MESSAGES

When an error has occurred in a program, correct the program referring to the "Checks" described in the following table.

Code	Item	Details

01	Error message
	Meaning
	Check
		Next Without For
		The NEXT statement does not correspond to a FOR statement.
		● Does the number of FOR statements match that of NEXT statement?
		● Is a branch taking place by GOTO or GOSUB outside the group in the FOR ~ NEXT loop?
		● Are OR ~ NEXT loops interlocked?
		(Example) `FOR I=1 TO 20`
		`∫`
		`FOR J=1 TO 20`
		`∫`
		`NEXT I`
		`∫`
		`NEXT J`

02	Error message Meaning Check	Syntax Error The instruction is not written in the correct syntax. ● Is a reserved word used for a variable name? ● Is a number greater than 63999 used as a Line No. ● Do the number and contents of instruction parameters match? ● Is an attempt made to assign a value to a reserved variable like ERL, ERR, or CSRLIN? ● Is the variable name correct? ● Are all essential parameters specified on the instruction?
03	Error message Meaning Check	Return Without Gosub The RETURN statement does not correspond with the GOSUB statement. ● Does a branch take place by other than a GOSUB statement within the subroutine?
04	Error message Meaning Check	Out Of Data No DATA statement to read. ● Does the number of variables to be input by the READ statement match the number of variables defined by the DATA statement? ● Are there enough DATA entries to execute the READ statement the required number of times? ● Has RESTORE been forgotten?
05	Error message Meaning Check	Illegal Function Call Naming of a function or statement is incorrect. ● Is the parameter specified in the correct range? ● Is the parameter format correct? ● Not enough arrays are specified (GET@). ● Is the array subscript negative?

Code	Item	Details
06	Error message Meaning Check	Overflow The operation result and input number are not in the allowable range. ● Is the integer operation result in the range from –32768 ~ 32767? ● Is the real number operation result in the range from -8.5×10^{37} ~ 8.5×10^{37}? ● Is the numeric obtained by integer conversion in the range from –32768 ~ 32767?
07	Error message Meaning Check	Out Of Memory The memory capacity is not large enough. ● Is the program too long? ● Is the DIM statement array too large?
08	Error message Meaning Check	Undefined Line Number The program line required does not exist. ● Is the Line No. specified by an instruction in the program?
09	Error message Meaning Check	Subscript Out Of Range The array subscript is not in the range specified by the DIM statement. ● Is the subscript 10 or greater in an array for which no DIM statement is used? ● If a variable is used for a subscript, is its value within allowable range? ● Does the number of dimensions of the DIM statement match the number of dimensions of the array variable?
10	Error message Meaning Check	Duplicate Definition An array and FN function are defined more than once. ● Is the execution of the same array declaration or FN function attempted twice during the program's execution?
11	Error message Meaning Check	Division By Zero Division using 0 as the divisor was attempted. ● Is the variable used as the divisor 0? ● Has the variable to be used as the divisor been defined?
12	Error message Meaning Check	Illegal Direct While waiting for a command, an attempt was made to execute an instruction that could not be used as a command.
13	Error message Meaning Check	Type Mismatch The data types do not match. ● Do the types (character type and numeric type) of left and right sides of the expression match? ● Do the types of arguments of a function match?
14	Error message Meaning Check	Out Of String Space The character area is not large enough. ● Has a large enough character area been reserved by a CLEAR statement?
15	Error message Meaning Check	String Too Long The number of characters in the character expression exceeds 255. ● Is the number of characters in the character expression operation greater than 255? ● Is the number of characters in the keyboard input greater than 255?

Code	Item	Details
16	Error message Meaning Check	String Formula Too Complex The character expression is too complicated. ● Is the nesting of parentheses too deep? ● Is the character expression too complicated because of too many parentheses?
17	Error message Meaning Check	Can't Continue The program execution cannot be continued by the CONT command. ● Has the program been changed after stopping using a STOP statement or by pressing Break ?
18	Error message Meaning Check	Undefined User Function An attempt was made to refer a FN function that has not been defined by a DEF FN statement.
19	Error message Meaning Check	No Resume No RESUME is given to the error processing routine. ● Is the error processing routine terminated by RESUME or END?
20	Error message Meaning Check	Resume Without Error A RESUME statement was executed while no error occurred. ● Erroneous branching to the error processing routine?
21 26~49 77~255	Error message Meaning	Unprintable Error An error to which no error message is defined has occurred.
22	Error message Meaning	Missing Operand The necessary parameter is not specified in the statement.
23	Error message Meaning Check	For Without Next The FOR and NEXT statements do not correspond. ● Does the number of FOR statements match the number of NEXT statements?
24	Error message Meaning Check	WHILE Without WEND The WHILE and WEND statements do not correspond. ● Does the number of WHILE statements match the number of WEND statements?

25	Error message
	Meaning
	Check
		WEND Without WHILE
		There is no WHILE statement to correspond an WEND statement.
		● Is a WEND statement executed without executing a WHILE statement?
		● Do the WHILE ~ WEND loop and FOR ~ NEXT loop cross?
		(Example)
		`┌FOR I=1TO 20`
		`│ ~`
		`┼WHILE A=1`
		`│ ~`
		`└NEXT I`
		`│ ~`
		`└WEND`

Code	Item	Details
50	Error message Meaning Check	Bad File Number The File No. is not in the range 1 ~ 16. ● If the File No. is a variable, is it in the range 1 ~ 16?
51	Error message Meaning Check	Bad File Mode The file mode is incorrect. ● Is an attempt being made to execute an I/O instruction not corresponding the mode at the time of OPEN? ● Is the parameter of 「mode」 in the OPEN statement written in small letters? (If it is, write it in capital letters.)
52	Error message Meaning Check	File Already Open Reopening of a file that is already opened was attempted. ● Was reopening attempted while a CLOSE statement was not used before? ● Is OPEN given for a File No. already used?
53	Error message Meaning Check	Device I/O Error An error occurred on a device being used. ● The cassette recorder does not operate with MB-S1. If the volume and quality of sound of the cassette recorder can be changed, change them and input again.
54	Error message Meaning Check	Input Past End An input instruction (INPUT#, LINE INPUT#) was executed after reading all data of the file. ● Was an attempt made to execute an input instruction without executing EOF?
55	Error message Meaning Check	Bad File Descriptor There is a format error in the file descriptor. ● Is the file name written correctly?
56	Error message Meaning Check	Direct Statement In File An attempt was made to load an instruction without a Line No. ● Is loading of data file being attempted?
57	Error message Meaning Check	File Not Open An attempt was made to execute an I/O instruction to a file that is not opened. ● Does the File No. of I/O instruction match that of the OPEN instruction?
58	Error message Meaning Check	Bad Data In File The type of data on the file does not match the type of data in the input instruction. ● Does the data type for output match the data type input?
59	Error message Meaning Check	Device In Use OPEN was applied to a device being used. ● Is OPEN applied to a cassette being used?

Code	Item	Details
60	Error message Meaning Check	Device Unavailable The I/O device is not in the condition able to input/output. Is a device name not permitted by the OPEN statement being used? (Pay particular attention to an error between 0 (zero) and O (Capital O.)
61	Error message Meaning Check	Buffer Overflow The I/O buffer is overflowing. Probably the speed of fetching data from the buffer by a program is too slow during use of RS-232C. If this is the case, set a slower baud rate.
62	Error message Meaning	Protected Program Execution, writing or correction of a command was attempted to a protected program.
– (Note 1)	Error message Meaning Check	Address Error (S1 BASIC) A non-existing address was referred to or writing to such an address was attempted. Probably referencing or writing to an unallocated address by MON is attempted while the user area has not been allocated. Is a graphic instruction used with NEW ON1 or NEW ON3?
– (Note 2)	Error message Meaning Check	System Call Error (S1 BASIC) An error occurred in a system call. Is system call being used correctly?

Note 1) Error codes 63 ~ 76 have been defined by S1 BASIC.

Note 2) These errors are unrecoverable. When an error of this type occurs, execution is aborted, and processing cannot be restarted by CONT or error processing by an ON ERROR routine.

4. LIST OF INSTRUCTIONS CLASSIFIED BY FUNCTION

Program preparation

- AUTO
- DELETE
- EDIT
- LIST
- LLIST
- MERGE
- NEW
- REM
- RENUM
- UNLIST

Program Load/Save/Execution

- CONT
- LOAD
- LOAD?
- LOADM
- RUN
- SAVE
- SAVEM

Declaration/Definition

- CLEAR
- DATA
- DEF FN
- DEF INT/SNG/DBL/STR
- DEF MAP
- DEF USR
- DIM
- NEW ON
- RESTORE

Program flow control

- END
- FOR~NEXT
- GOSUB~RETURN
- IF~THEN~ELSE
- ON~GOSUB
- ON~GOTO
- RETURN
- RUN
- STOP
- WHILE~WEND

Data assignment

- LET
- LSET/RSET
- MID$
- READ
- SWAP

Character string operation

- INSTR function
- LEFT$ function
- LEN function
- LOF function
- MID$
- MID$ function
- POS function
- RIGHT$ function
- SPACE$ function
- SPC function
- STRING$ function

Data conversion

- ASC function
- CDBL function
- CHR$ function
- CINT function
- CSNG function
- CVI, CVS, SVD function
- HEX$ function
- MKI$, MKS$, MKD$ function
- OCT$ function
- STR$ function
- VAL function

Arithmetic calculation

- ABS function
- ATN function
- COS function
- EXP function
- FIX function
- FN function
- INT function
- LOG function
- RANDOMIZE function
- RND function
- SGN function
- SIN function
- SQR function
- TAN function

Interrupt

- COM (n) ON/OFF/STOP
- KEY (n) ON/OFF/STOP
- MTRIG (n) ON/OFF/STOP
- STRIG (n) ON/OFF/STOP
- ON COM (n) GOSUB
- ON ERROR GOTO
- ON INTERVAL GOSUB
- ON KEY (n) GOSUB
- ON MTRIG (n) GOSUB
- ON STRIG (n) GOSUB

Error processing

- ERR function
- ERL function
- ERROR
- ON ERROR GOTO
- RESUME

Text screen processing

- CLS
- COLOR
- CONSOLE
- LCOPY
- LIST
- LOCATE
- PALETTE
- PRINT
- PRINT USING
- SCREEN
- WIDTH
- CSRLIN function
- POS function
- SCREEN function
- TAB function

Graphic screen processing

- CIRCLE
- CLS
- COLOR
- CONNECT
- CONSOLE
- GET@
- GCURSOR
- KANJI
- LCOPY
- LINE
- PAINT
- PALETTE
- POINT
- PRESET
- PSET
- PUT@
- SCREEN
- SYMBOL
- VIEW
- WINDOW
- MAP function
- POINT function
- VIEW function
- WINDOW function

Debug processing

- MON
- TRON/TROFF

Cassette tape

- CLOSE
- FILES
- INPUT #
- LFILES
- LINE INPUT #
- MOTOR
- OPEN
- SKIPF
- EOF function
- POS function

Keyboard

- CLOSE
- INPUT
- INPUT WAIT
- INPUT #
- LINE INPUT
- LINE INPUT #
- OPEN
- INKEY$ function
- INPUT$ function

Printer

- CLOSE
- LCOPY
- LFILES
- LLIST
- LPRINT, LPRINT USING
- OPEN
- PRINT # (USING)
- POS function

Communications line

- CLOSE
- COM (n) ON/OFF/STOP
- INPUT #
- LINE INPUT #
- OPEN
- TERM
- EOF function
- POS function

Mouse

- MTRIG (n) ON/OFF/STOP
- ON MTRIG (n) GOSUB
- MREAD function
- MTRIG function

Joystick

- STRIG (n) ON/OFF/STOP
- ON STRIG (n) GOSUB
- STICK function
- STRIG function

Sound

- BEEP
- PLAY
- SOUND

Image generator

- IG$
- IMAGE

Machine language processing

- DEF USR
- EXEC
- MON
- POKE
- FRE function
- PEEK function
- USR function
- VARPTR function

Time processing

- INTERVAL
- INTERVAL ON/OFF/STOP
- DATE function
- DATE$ function
- ON INTERVAL GOSUB
- TIME function
- TIME$ function

Function key processing

- KEY
- KEY LIST
- KEY (n) ON/OFF/STOP
- ON KEY (n) GOSUB